VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA ELEKTROTECHNIKY A KOMUNIKAČNÍCH TECHNOLOGIÍ
FACULTY OF ELECTRICAL ENGINEERING AND COMMUNICATION

ÚSTAV MIKROELEKTRONIKY
DEPARTMENT OF MICROELECTRONICS

NÁVRH, ANALÝZA A IMPLEMENTACE POKLADNÍHO SYSTÉMU
DESIGN, ANALYSIS AND IMPLEMENTATION OF THE CASH REGISTER SYSTEM

DIPLOMOVÁ PRÁCE
MASTER'S THESIS

AUTOR PRÁCE — **Bc. Mikuláš Ponechal**
AUTHOR

VEDOUCÍ PRÁCE — **doc. Ing. Lukáš Fujcik, Ph.D.**
SUPERVISOR

BRNO 2020

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
FAKULTA ELEKTROTECHNIKY A KOMUNIKAČNÍCH TECHNOLOGIÍ

Diplomová práce

magisterský navazující studijní obor **Mikroelektronika**

Ústav mikroelektroniky

Student: Bc. Mikuláš Ponechal ***ID:*** 186167
Ročník: 2 ***Akademický rok:*** 2019/20

NÁZEV TÉMATU:

Návrh, analýza a implementace pokladního systému

POKYNY PRO VYPRACOVÁNÍ:

Cílem diplomové práce je návrh a implementace integrační vrstvy pokladního systému, ktorý bude v pilotním provozu ověřen v Zoo Brno. Základem práce je seznámení se s obecnými principy pokladního systému a analýza souvisejícího pokladního HW. Samotná práce spočíva v analýze interakce pokladního systému s HW, detailním návrh integračních mechanismů a rozhraní a následném vytvoření prototypu ve zvoleném programovacím jazyku.

DOPORUČENÁ LITERATURA:

Podle pokynů vedoucího práce.

Termín zadání: 3.2.2020 ***Termín odevzdání:*** 1.6.2020

Vedoucí práce: doc. Ing. Lukáš Fujcik, Ph.D.

doc. Ing. Lukáš Fujcik, Ph.D.
předseda oborové rady

ABSTRAKT

Diplomová práca uvádza do problematiky pokladničných systémov používaných v Brnenskej zoo a v Starezsport Brno. Cieľom práce bolo naštudovať spomínané systémy a používaný hardvér, analyzovať, navrhnúť a vytvoriť vlastný pokladničný systém. Prvá kapitola zahŕňa rozbor pokladničných systémov a prehľad vybraných technológií. Druhá je uvádza návrhy, postupy pri vývoji. V tretej kapitole sa nachádza stručná dokumentácia k používaniu systému a posledná, štvrtá kapitola je venovaná testovaniu základných funkcionalít celého systému.

KĽÚČOVÉ SLOVÁ

Pokladničný systém, turniket, užívateľské rozhranie, QR kód, Raspberry Pi.

ABSTRACT

The masters's thesis introduces cash register system currently used by Brno zoo and Starezsport Brno. The aim of the thesis was to study and describe above-mentioned systems and hardware they are interacting with, to analyze, design and create new custom cashier system. The first chapter includes the analysis of cashier systems, and overview of selected technologies for developement. Second chapter introduces developement designs and practices. The third chapter contains short system documentation and the last, fourth chapter is devoted to testing basic cashier system functionality.

KEYWORDS

Cashier system, turniquet, user interface, QR code, Raspberry Pi.

PONECHAL, Mikuláš. *Návrh, analýza a implementace pokladního systému.* Brno, 2020. Dostupné také z: https://www.vutbr.cz/studenti/zav-prace/detail/127420. Diplomová práce. Vysoké učení technické v Brně, Fakulta elektrotechniky a komunikačních technologií, Ústav mikroelektroniky. Vedoucí práce Lukáš Fujcik.

POĎAKOVANIE

Ďakujem vedúcemu diplomovej práce doc. Ing. Lukášovi Fujcikovi, Ph.D. za jeho čas venovaný konzultácii diplomovej práce, za rady pre jej zlepšenie. Ďalej ďakujem konzultantovi Mgr. Petrovi Halmovi a ďalším kolegom zo spoločnosti inQool a.s, za pomoc a usmernenie pri vypracovaní tejto diplomovej práce.

OBSAH

ÚVOD

Pokladničné systémy sú neoddeliteľnou súčasťou fungovania podnikov, v ktorých dochádza k predaju či už tovaru alebo služieb. Ich hlavnou úlohou je evidovať tržby, poskytovať prevádzkarom prehľad o vývoji podniku.

Cieľom diplomovej práce je vytvoriť analyzovať a navrhnúť nový pokladničný systém, ktorý bude schopný zaručiť rovnakú funkcionalitu rozšírenú o požiadavky, ktoré terajšie systémy v analyzovaných spoločnostiach neponúkajú.

Prvá kapitola sa zaoberá analýzou používaných pokladničných systémov v Brnenskej zoo a v Starezsport v Brne. Analýza je venovaná ako softvérovej časti systému, tak aj hardvérovým zariadeniam, ktoré sa používajú. Zároveň je tu rozobraná funčná stránka daných zariadení a ich spôsob komunikácie s pokladničným systémom. Záver kapitoly obsahuje prehľad vybraných technológií, ktoré budú použité pri realizácií vlastného systému.

V druhej kapitole sa nachádza popis použitých postupov pri plnení implementovaní vlastného pokladničného systému, v ktorom už nie je použitý pôvodný koncept obsahujúci middleware komponent z dôvodov, ktorých výsledkom bolo získanie nedostatočných informácií potrebných pre realizáciu tohto postupu. Ďalej sa v kapitole nachádza aj vizuálny návrh užívateľského rozhrania, opis jeho častí, z ktorých je zložené, charakteristika serverovej časti aplikácie a opis hardvéru slúžiaceho na skenovanie QR kódov a komunikáciu so serverom pokladničného systému.

Tretia kapitola poskytuje dokumentáciu k pokladničnému systému a opisuje spôsoby jeho používania, zoznamuje s jednotlivými procesnými krokmi.

Posledná, štvrtá kapitola sa venuje testovaniu základných funkcií pokladničného systému. Jej cieľom je dokázať, že aplikáciu je možné ovládať prostredníctvom užívateľského rozhrania, otestovať proces komunikácie užívateľského rozhrania so serverom, použitého hardvéru so serverom a jeho správnu funkcionalitu pri procese skenovania vstupeniek.

1 ZOZNÁMENIE SA POKLADNIČNÝM SYSTÉMOM

Táto kapitola je venovaná rozboru pokladničného systému, ktorý sa aktuálne používa v Brnenskej zoo a v spoločnosti Starezsport Brno. Oba systémy sa skladajú z dvoch hlavných častí - časti fyzickej (hardvérovej) a časti aplikačnej (softvérovej), pričom obe časti sú od seba závislé. Každý systém má svoje špecifiká podľa požiadaviek, ktoré si inštitúcie stanovili pred ich implementáciou.

1.1 Konfigurácia pokladničného systému v zoo

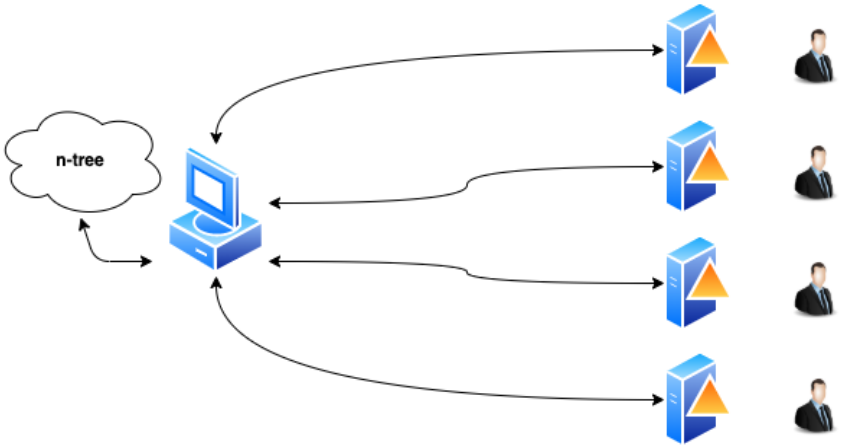
Softvérovým riešením v Brnenskej zoo je pokladničný systém s názvom n-tree. Ide o komplexný softvér vytvorený rakúskou spoločnosťou n-tree solutions, ktorý sa aktuálne používa. Tento software bol navrhnutý za účelom zjednodušenia a zefektívnenia manažovania návštevníckych systémov v dvoch oblastiach:

- v oblasti pre správu návštevníkov,
- v oblasti pre správu destinácií (kartové platformy) [1].

Systém v zoo Brno spadá pod prvú oblasť, t.j. implementácia systému je so zameraním na správu návštevníkov. Softvér zabezpečuje komunikáciu fyzických zariadení s centrálnou pokladňou, zabezpečuje precíznu kontrolu všetkých pokladničných operácií, overovanie platnosti lístkov ako aj spracovanie informácií, potrebných na vyhodnocovanie štatistických údajov ako napr. prehľad o príjmoch, návštevnosti za určité obdobie, a pod. Dodávateľom systému do Brnenskej zoo je firma NESSY, s.r.o [2].

Z pohľadu parametrov je systém schopný spracovať do 3600 vstupenkových kariet za hodinu, pričom odbavovací systém disponuje (prístroje so snímačmi a synchrónnymi motormi - turnikety, vstupné brány) priepustnosťou do 1200 osôb za hodinu [2].

Pokladňa je de facto klasický počítač, vybavený už spomínaným softvérom n-tree, ktorý je kľúčovým prvkom celého systému a je umiestnený do jednej z pokladní. Pokladňa osobitne komunikuje s odbavovacími zariadeniami, konkrétne s tromi kusmi turniketov typu DOLOMIT, a jednou vstupnou bránkou typu IKARUS BADEN, ktorá plní svoj účel pre vstup s kočíkmi alebo pre ľudí s hendikepom. Topológia systému je znázornená na obr. 1.1.



Obr. 1.1 – pokladničný systém v zoo Brno

Automatické odbavovacie zariadenie sa skladá z pohyblivej časti, ktorú tvoria motorom poháňané ramená v prípade turniketu alebo motorom poháňané dvierka v prípade bránky, a snímač kariet TRX 100-IR s technológiu čiarového kódu, ktorého súčasťou je aj dvojriadkový displej slúžiaci na zobrazovanie informácií pre samotného zákazníka, ako napr. počet zostávajúcich vstupov alebo dôvod zamietnutia vstupu a skener pre čítanie relevantných informácií z čipu bezkontaktnej karty alebo čiarového kódu klasickej karty.

Odbavovací systém nevyhodnocuje informácie z karty sám. Získané informácie sa vyhodnocujú až po doručení na riadiaci pokladničný PC, po vyhodnotení dáta putujú späť do odbavovacieho systému, kde sa výsledok zobrazí zákazníkovi. Pokladničný PC slúži zároveň aj na tlačenie samotných vstupenkových kariet priamo pri predaji na jednej z tlačiarní od firmy Intermec. Vstupenkové karty sú kódované, kódovanie je realizované programovo, prostredníctvom programovacieho modulu cez rozhranie RS-232. Používané sú dva typy kariet - karty s čiarovým kódom a bezkontaktné čipové karty [2].

Výhoda kariet s čiarovým kódom je, že kódovanie je lacnejšie, sú šetrnejšie k životnému prostrediu, zaisťujú "zabezpečenie" proti strate dát vplyvom teploty, mechanického namáhania (lámania) alebo magnetického poľa [2].

Bezdotykové chipkarty na rozdiel od klasických, už spomínaných kariet s čiarovým kódom, majú pamäťový prvok, vďaka ktorému sú schopné prenášať aj dynamicky sa meniace informácie. Tieto karty taktiež umožňujú zavedenie súbežnej evidencie o karte a to priamo v samotnom čipe karty zároveň v systéme n-tree. Výhodou (z užívateľského hľadiska) je tiež fakt, že bezdotykovú kartu stačí iba priložiť do priamej blízkosti snímača kariet, zatiaľ čo kartu s čiarovým kódom je nutné zasunúť dovnútra integrovaného snímača (cca 1,5 cm), aby ju dokázal integrovaný skener správne prečítať [2].

1.1.1 Turniket Dolomit

Jedná sa o motorizované zariadenie od firmy GOTSCHLICH. Tento typ turniketu je vyrobený z pozinkovanej ocele a nerezovej ocele, čo ho robí efektívnym najmä pre vonkajšie použitie, t.j. pre zabezpečenie vstupov do outdoorových areálov. Má dve prevedenia, a to dvojramenné a trojramenné, pričom je možné čítačky vstupenkových kariet ľubovoľne umiestniť a pripojiť pomocou nastaviteľných držiakov.

V klasickej prevádzke turniket funguje tak, že rotačná hviezdica (ramená turniketu) je držaná prostredníctvom elektronickej brzdy v uzamknutej polohe. Brzda je uvoľnená snímacím systémom vstupenkových kariet, a to vždy pri vyhodnotení platného vstupu. Tým sa prechod turniketom uvoľní v požadovanom smere [3].

Turniket tiež v sebe obsahuje vnútorné integrované senzory, ktoré zabraňujú zraneniam prechádzajúcich osôb. Trecia spojka turniketu umožňuje jeho otvorenie aj v uzamknutom stave vyvinutím sily viac ako 40 kN. V prípade, že je brzda vypnutá, je možné rotačnú hviezdicu otočiť aj za použitia malej sily. Vo verzii, kedy má turniket len dve ramená existuje poloha, kedy žiadne rameno neblokuje cestu, teda iba ako voľný prechod, resp. východ [3].

1.1.2 Bránka IKARUS Baden

Je zariadenie od rovnakej firmy ako turniket Dolomit. Jedná sa o motorizovanú bránku, ktorá rozširuje funkcionalitu turniketov umožnením vstupu pre detské kočiare a invalidné vozíky. Je vyrobená výhradne z nehrdzavejúcich materiálov, preto je vhodná aj pre plavecké rekreačné zariadenia. Môže slúžiť ako jednosmerný alebo obojsmerný priechodzí bod [4].

Snímač od firmy n-tree solutions, slúži na čítanie vstupenkových kariet, čím riadi a zároveň zabezpečuje prístup do areálov prostredníctvom odblokovania turniketu/bránky na základe informácií získaných po zosnímaní zákazníkovej vstupnej karty. Vstavaný optický skener na čiarové kódy s vysokým rozlíšením dokáže prečítať čiarový kód dvestokrát za sekundu.

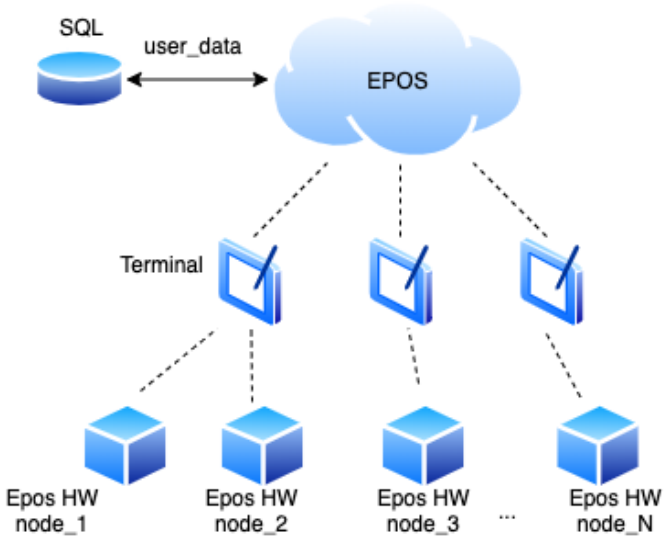
Pridaním RFID technológie snímač dokáže prečítať bezkontaktnú chipkartu do vzdialenosti 20 cm [5].

1.2 Konfigurácia systému v Starezsport Brno

Starezsport používa pokladničný systém EPOS. Skratka EPOS znamená Electronic Point of Sale system, teda je to elektronický systém na spracovanie, sledovanie predaja a výkonnosti jednak podniku ako celku, ale aj zamestnancov osobitne. Organizačné jednotky vytvárajú stromovú štruktúru systému. Poskytovateľom systému do Starezsport v Brne je firma HASAM. Hierarchia systému je znázornená na obr. 1.2.

Existuje viac druhov EPOS softvéru v závislosti na druhu podnikania. Základné funkcie EPOS systému:

- riadenie a sledovanie zásob,
- nákup a spracovanie objednávok,
- vernostné programy akcie a zľavy,
- integrácia s e-commerce [6].



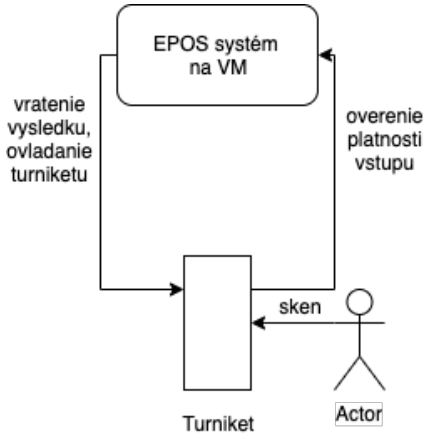
Obr. 1.2 - hierarchia systému EPOS

Centrálny terminál (pokladňa) komunikuje so službou bežiacou na serveri/cloude prostredníctvom HTTP protokolu, bez nutnosti prístupu jednotlivých staníc k súborovej štruktúre servera. Údaje o užívateľoch sú uložené v centrálnej SQL databáze, citlivé údaje sú, samozrejme, zašifrované. Terminál, pokladne, turnikety je možné spolu spojiť prostredníctvom internetu, ale iba v rámci rozsahu internej siete. EPOS je otvorený systém, teda sú prítomné kanály pre informačné systémy tretích strán, napr. pre webové predajné portály [7].

Hardvérová stránka EPOS môže obsahovať rôzne komponenty, ktorých prítomnosť v systéme sa odvíja od požiadaviek zákazníka. Hlavným prvkom je terminál, v ktorom sa zadávajú objednávky a spracúvajú platby. V posledných rokoch sa stal systém prístupnejším, nakoľko hardvérové zariadenia všeobecne zaznamenali cenový pokles. Medzi EPOS kompatibilné prídavné zariadenia patrí:

- kasa (zamykateľná kovová krabica na uchovávanie peňazí),
- tlačiareň,
- elektronický skener čiarových kódov,
- užívateľský display,
- PDQ terminál,
- tablet/ iPad EPOS,
- PC.

Ako zariadenia pre kontrolu vstupu sa používajú turnikety od firmy HASAM a COMINFO v kombinácií s multifunkčným terminálom REA::TICKET. Turnikety, bránky sú pripojené do internej siete fyzicky, štruktúrovanou kabelážou cez protokol Ethernet. Každé zariadenie má vlastný mikropočítač a sieťovú kartu s unikátnou IP adresou. V mikropočítači beží 32-bitový operačný systém WinCE verzie 6.0 [7].



Obr. 1.3 - ovládanie turniketu

Proces vstupu zákazníka sa začína oskenovaním jeho čipovej karty, lístka s bar/QR kódom (obr. 1.3). Snímač na turnikete získa údaje zo vstupenky, ktoré vzápätí pošle systému EPOS bežiacemu na VM s konkrétnou IP adresou. Tu sa overí platnosť vstupu, výsledok overenia putuje späť na príslušný turniket, tzn. turniket s rovnakou adresou, z ktorej prišla požiadavka na overenie. O obsluhu turniketu sa stará klient aplikácie EPOS bežiaci na už spomínanom OS WinCE. Hardvér turniketu je pripojený na virtuálny COM port slúžiaci na výmenu informácií s klientom aplikácie, pričom na ovládanie turniketu sa používa súbor `AT` príkazov.

1.2.1 Turniket HASAM TS-121-1014

Turniket používaný spoločnosťou Starezsport Brno. V rámci teoretického prieskumu nám bolo umožnené preskúmať jeden konkrétny turniket vo wellness oddelení. Turniket obsahuje dva mikropočítače (jeden pre ovládanie vstupu, druhý pre ovládanie výstupu) prepojené cez HUB do internej siete a motor pre ovládanie ramien. Má v sebe zabudovanú čítačku čipových kariet/barkódov, ktorá komunikuje so serverom, na ktorom beží EPOS. Povolenie alebo zamietnutie vstupu sú farebne odlíšené, osvetlenie turniketu riadi riadiaca LED jednotka, ktorá pracuje s 5 V logikou. Napájaný je zo siete, z bezpečnostných dôvodov celý turniket pracuje s napätím 24 VDC. Súčasťou turniketu je taktiež elektronická brzda, ktorá bráni násilnému prechodu cez turniket.

Terminál umožňujúci čítanie súčasne čiarových kódov , kódov QR (aj z obrazovky smartfónu), bezkontaktných kariet a tiež náramkov. Zariadenie disponuje možnosť dovybavenia internou pamäťou, ktorá slúži na overovanie vstupeniek v prípade zlyhania/výpadku servera. Typicky sa inštaluje na turnikety [8].

1.3 Windows CE

Operačný systém navrhnutý pre menej výkonné zariadenia akými sú mobilné telefóny, terminály, TV/video konzoly, pokladne, turnikety, atď. Vyvinutý bol v roku 1996, začínal verziou CE 1.0. V turniketoch sa používa verzia CE 6.0, vydaná v roku 2006, ktorá prešla oproti svojim predchodcom výraznými zmenami. Každý proces má teraz k dispozícii

2 GB virtuálnej pamäte namiesto pôvodných 32 MB. Od tejto verzie je možné inštalovať alebo vytvárať ovládače periférií [9].

Pre vývojárov existuje súbor API funkcií vychádzajúcich z klasickej Win32 API a zároveň aj API pre vstavané systémy. Momentálne verzia CE 6.0 podporuje štyri procesorové architektúry - ARM, MIPS, SH4 a x86. Systém ponúka využitie širokého spektra technológií, ako napr. [9]:

- .NET Compact Framework 2.0 a 3.5,
- podporu viacerých jazykov,
- TCP/IP, IPv4, IPv6 protokoly,
- PAN, WAN, LAN, Bluetooth, Wi-Fi,
- VoIP, RTC, RDP,
- FTP,
- Internet Explorer,
- CD File System.

Vývojárske nástroje pre Win CE 6.0 sú integrované vo vývojovom štúdiu Visual Studio 2005 od firmy Microsoft [9].

1.4 Vybrané technológie a zariadenia pre návrh nového pokladničného systému

Obsahom tejto podkapitoly je popis vybraných technológií, ktoré budú použité na vytvorenie pokladničného systému. Sú to konkrétne:

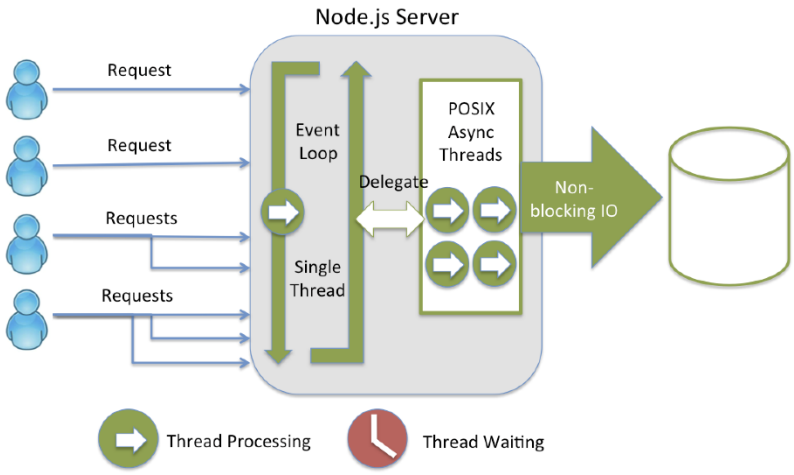
- node.js,
- react.js,
- rest API,
- QR kód
- Raspberry Pi

Súbor technológií bol vybraný najmä na základe osobných skúseností s jazykom JavaScript, z ktorého node.js a react.js priamo vychádzajú. Ďalšie dôvody sú rýchlosť, použiteľnosť takmer na každej platforme, ako aj dostupnosť informácií. REST API bude použité ako štandard komunikácie node.js servera s databázou.

1.4.1 Node.js

Asynchrónny JavaScript runtime Node.js je ako technológia používaný na vytváranie sieťových aplikácií, od ktorých sa očakáva obsluha mnohých súbežných akcií. Obsluha prichádzajúcich požiadaviek dodržiava tzv. “callback pattern”. Takmer žiadna z funkcií nevykonáva I/O operácie priamo, čím sa zaručí neblokujúci proces obsluhy akcií. Node.js využíva V8 engine, teda jadro z prehliadača Google Chrome. Klasický Javascript beží

v prehliadači, node.js priamo na zariadení, pričom na rozdiel od klasického JS má prístup k súborovej štruktúre [10,11,12]. Princíp fungovania je uvedený na obr. 1.4.



Obr. 1.4 - všeobecný model Node.js aplikácie [13]

Oproti technológiám ako sú napríklad PHP alebo Java má Node.js výhodu v operačnej rýchlosti, rýchlejšom osvojení programátorom. Jedná sa o modernú technológiu, ktorá je vhodná pre projekty menších rozmerov [13,14].

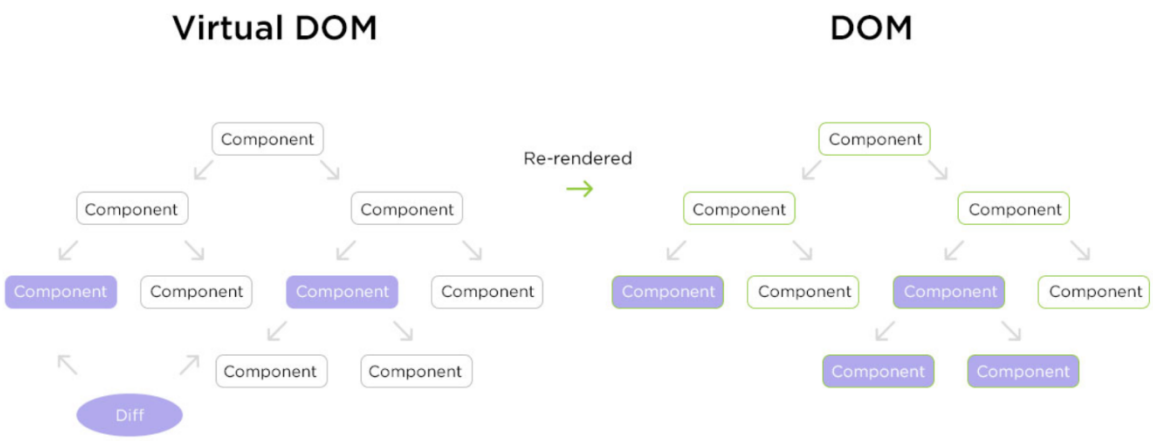
1.4.2 React.js

Javascriptová knižnica React.JS bola vyvinutá spoločnosťou Facebook a je aktuálne populárna v svete web vývoja, budovania užívateľských rozhraní a mobilných aplikácií, resp. frontendovej časti webových aplikácií. Jazyk používa tagovanú JSX syntax, ktorá na prvý pohľad môže pripomínať šablónovací jazyk, napr. HTML. Na preloženie JSX syntaxe do JavaScriptu sa používa Babel compiler. Ukážka sa nachádza na obr. 1.5 [15].

```
const element = <h1>Hello, world!</h1>;
```

Obr. 1.5 - syntax React.js [15]

Na rozdiel od klasických prehliadačových DOM elementov sú react elementy prosté objekty. React komponenty, elementy sa väčšinou vykresľujú do koreňového elementu s id “root”. Koreňového preto, lebo v jeho vnútri bude všetko manažované cez React DOM. Pretože React porovnáva elementy s predchádzajúcim stavom (pred tým, ako bola užívateľom vyvolaná zmena), v užívateľskom rozhraní sa prekresľujú iba uzly, v ktorých došlo naozaj k zmene obsahu (obr. 1.6), s cieľom dostať aplikáciu do vopred stanoveného a požadovaného stavu [16].



Obr. 1.6 - princíp prekresľovania React.js [16]

Výhodou používania React.js je hlavne voľnosť, ktorú poskytuje pri vytváraní užívateľských rozhraní alebo aplikácií. Ďalším faktorom je rýchlosť. Spolu s rozrastajúcou sa komunitou react programátorov sa zväčšuje aj dosah tejto technológie.

1.4.3 REST

Ide o architektonický štýl navrhovania API - skratka pre aplikačné programovacie rozhranie. Toto rozhranie umožňuje kontaktovať externú službu/server s použitím preddefinovaných súborov príkazov. Rozhranie sa rozhraním nazýva preto, lebo práve na tomto mieste sú spolu v interakcii rôzne odlišné softvérové komponenty s cieľom splniť účel danej problematiky.

Termín REST znamená **RE**presentational **S**tate **T**ransfer, pričom ide o návrhový štýl pre distribuované systémy médií. Hlavné zásady REST systémov:

1. **Oddelenie klientskej a serverovej časti systému** – dochádza tým k odstraňovaniu problémov pri ukladaní dát a zlepšuje sa škálovateľnosť systému.
2. **Bezstavovosť** - každá požiadavka od klienta musí byť zadaná platným spôsobom, musí obsahovať všetky potrebné parametre stanovené pri návrhu API.
3. **Možnosť uloženia odpovede do pamäte cache**, dáta môžu byť uložené do pamäte cache podľa toho, ako sú označené v odpovedi. Ak sú dáta označené ako cacheovateľné, klient si ich môže uložiť do pamäte cache a použiť ich neskôr pri ekvivalentných žiadostiach.
4. **Jednotné rozhranie**, ktoré je dosiahnuté vymedzením obmedzení správania REST komponentov.
5. **Prítomnosť systémových vrstiev.** Vrstvením systému sa dosahuje členenie na menšie hierarchické celky s požadovaným obmedzením - komponenty v jednotlivých vrstvách už nevidia, čo sa deje vo vrstve za komponentom, s ktorým sú vo vzájomnej interakcii.

Kľúčovou abstrakciou informácie, ktorú chceme získať prostredníctvom REST API je prostriedok. REST používa identifikátory týchto prostriedkov pre rozpoznanie konkrétneho zdroja zapojeného do interakcie medzi komponentami. Každý prostriedok má príslušnú reprezentáciu a tá má svoj formát.

RESTful API umožňuje vytvorenie aplikácie so všetkými CRUD (create, retrieve, update, delete) operáciami. Na vykonávanie operácií sa používajú metódy HTTP - GET, POST, PUT, DELETE [17].

1.4.4 QR kód

Jedná sa o 2D typ čiarového kódu. Rozdiel medzi 2D a 1D kódmi spočíva v spôsobe skenovania dát a kapacity obsahu dát. Zatiaľ čo 1D kódy sa skenujú iba v horizotálnom smere (lineárne), 2D kódy nesú informácie v horizontálnom a vertikálnom smere súčasne (maticovo). Klasické 1D čiarové kódy prenášajú okolo 20 digitov, QR kód verzie 40 až do 4,296 znakov, pričom finálna kapacita QR kódu je závislá aj na parametre ECC – error correction feature, ktorý udáva, koľko dát je schopný QR kód „opraviť“ ak je príslušný QR kód poškodený alebo znečistený. Sú 4 úrovne ECC [18,19]:

1. úroveň L (približne 7%),
2. úroveň M (približne 15%),
3. úroveň Q (približne 25%),
4. úroveň H (približne 30%),

t.j. čím je väčšia úroveň ECC tým má QR kód menšiu kapacitu. Úroveň sa volí podľa prostredia, v ktorom sa s kódom manipuluje. Najčastejšie sa používa variant M, Q a H sa používajú na miestach, kde môže prísť k poškodeniu kódu ľahšie napr. vo veľkých výrobách. Úroveň L sa používa v opačnom prípade – v čistých priestoroch prípadne pri potrebe prenášať väčší objem dát [18,19].

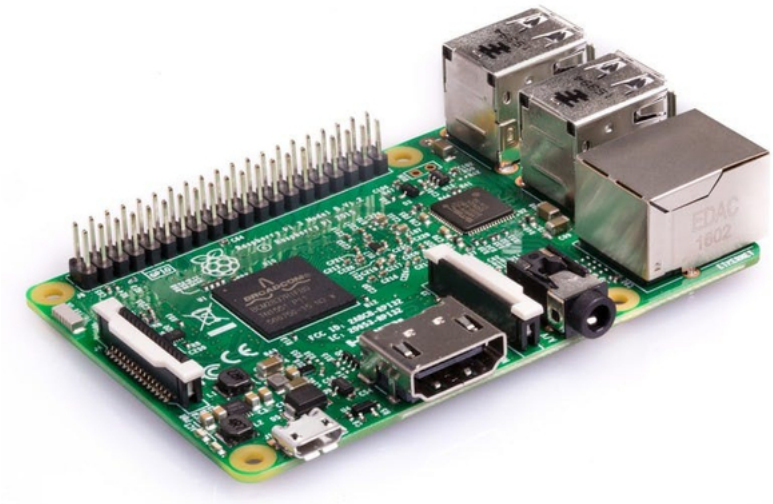
Obr. 1.7 - QR kód

1.4.5 Raspberry Pi

Raspberry Pi je známy embedded system - jednodoskový počítač o veľkosti kreditnej karty, na ktorom beží natívne Linux based systém Raspbian, ale dokáže na ňom bežať v podstate akýkoľvek operačný systém, ktorý sa bootuje z SD karty. Používa procesor architektúry ARM. Súčasne existuje päť verzii Raspberry Pi [20]:

1. **Raspberry Pi Zero (Zero W)** – disponuje jednojadrovým 1 GHz procesorom, 512 MB RAM pamäťou, mini HDMI portom, CSI konektor na kameru.
2. **Raspberry Pi 1** – základný model disponuje jednojadrovým 0,7 GHz procesorom, 512 MB RAM, 2x USB portami, 1x HDMI port, Ethernet port, 40 GPIO pinmi.
3. **Raspberry Pi 2** – základný model ponúka 4x USB porty, 40 GPIO pinov, HDMI port, Ethernet port, CSI rozhranie, DSI rozhranie (display), VideoCore IV 3D grafické jadro.
4. **Raspberry Pi 3** – základný model využíva štvorjadrový 1.2 GHz 64 bitový Broadcom BCM2837 procesor, 1 GB RAM, BCM43438 bezdrôtový BLE modul, 40 GPIO pinov, 8x USB portov, CSI, DSI rozhrania, 1x HDMI.
5. **Raspberry Pi 4** – posledná najnovšia verzia, využívajúca BCM2711, štvorjadrový Cortex-A72 64 bitový 1.5 GHz procesor, až do 4 GB LPDDR4-3200 SDRAM, 2.4 a 5 GHz Wi-Fi a BLE, gigabitový Ethernet, 2x micro HDMI, OpenGL ES 3.0 grafiku, USB-C konektor.

Každá z verzií má rôzne variácie (modely), pričom jednotlivé verzie sa najčastejšie líšia počtom GPIO pinov, USB portov, veľkosťou celého embedded systému a podobne [20].



Obr. 1.8 - Raspberry Pi model 3B [21]

1.4.6 Sieťové technológie

Aby implementácia čítacieho zariadenia vstupeniek mala zmysel, je nutné zabezpečiť jeho komunikáciu so serverom. Server, na ktorom pobeží API sa bude nachádzať v sieti (lokálnej/vonkajšej) a z tohto dôvodu vyplýva nutnosť začlenenia čítacieho zariadenia do siete, v tomto prípade do lokálnej siete pripojením k smerovaču. Existujú dva najjednoduchšie spôsoby ako realizovať pripojenie skenovacieho zariadenia k sieťovému smerovaču:

1. fyzickým médiom,
2. bezdrôtovo.

Pri návrhu netreba dbať na elektrickú spotrebu zariadenia, pretože sa počíta

s prípadom, kedy bude zariadenie napájané z elektrickej siete. Preto je možné použiť bez obmedzenia technológie ako Wi-Fi a Ethernet. Ich parametre sú závislé na poskytovateľovi internetových služieb, ktorý bude poskytovať internetové pripojenie danej inštitúcií.

Skratka Wi-Fi označuje technológiu bezdrôtového LAN (WLAN) prenosu dát podľa štandardu 802.11 IEEE normy. Dáta sa prenášajú najčastejšie v pásme 2,4 GHz (802.11g), pričom prenosová rýchlosť dosahuje maximálne 54 megabitov za sekundu. Novším štandardom je 802.11ac, ktorý používa prenosové pásmo 5 GHz s potenciálom dosiahnuť prenosovú rýchlosť do 1,3 gigabitov za sekundu. V súčasnosti sa vyvíja ďalší štandard a to 802.11ad, ktorý by mohol operovať s prenosovým pásmom 60 GHz s teoretickým maximom prenosovej rýchlosti dosahujúcej 7 gigabitov za sekundu [25, 26].

Ethernet je technológia spájajúca zariadenia najčastejšie v lokálnej (LAN) sieti využívajúca fyzické médium na prenos dát. IEEE špecifikovala súhrn štandardov 802.3, ktorými je Ethernet charakterizovaný. Najrýchlejší Ethernet v dnešnej dobe disponuje maximálnou prenosovou rýchlosťou do desať gigabitov za sekundu [24].

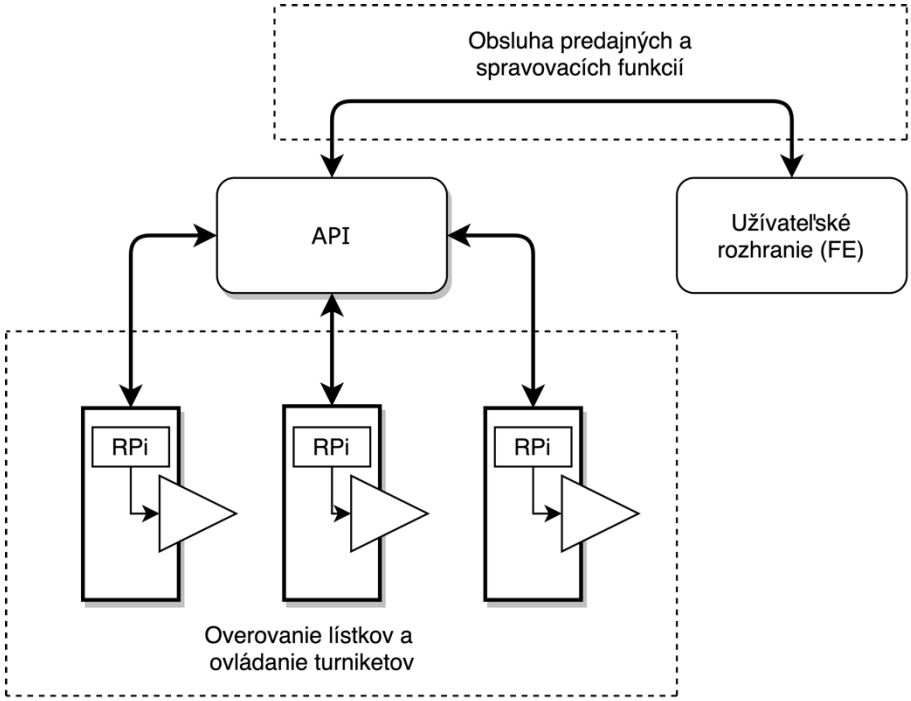
V porovnaní s Wi-Fi je výhodou väčšia spoľahlivosť pripojenia, väčšia prenosová rýchlosť, menšia odozva v sieti, menšia chybovosť prenosu. Medzi nevýhody patrí nutná prítomnosť fyzického média (káblu) spôsobujúca menšiu mobilitu zariadení komunikujúcich prostredníctvom Ethernet [24, 25, 26].

2 NÁVRH POKLADNIČNÉHO SYSTÉMU

Táto kapitola je venovaná návrhu pokladničného systému, ktorý bude univerzálny pre typy podnikov, ktoré majú fungovanie prevádzok založené na princípe uvedených inštitúcií v kapitole 1, prípadne pre akékoľvek ďalšie podniky s cieľom ich "zjednotenia" a umožnenia vylepšenia systému v oblasti poskytovania služieb, prípadnej podpory novej funkcionality. Zjednotením sa v tomto prípade myslí stav, kedy si užívateľ bude môcť kúpiť, lístok, resp. vykonať ľubovoľnú transakciu z jedného miesta do ktoréhokoľvek podniku využívajúceho tento pokladničný systém, pričom bude zaručená všade jednotná funkcionalita systému. Implementácia pokladničného systému má ďalej zaručiť aj lepšiu infraštruktúru, prehľadnosť, ovládateľnosť a to najmä z pohľadu zamestnancov. V rámci diplomovej práce bude vytvorený koncept pokladničného systému – užívateľské rozhranie, logika zabezpečujúca chod a funkciu celého systému ako aj čítačku vstupeniek. Koncept vytvorený v rámci tejto práce sa nemusí zhodovať s produkčnou verziou.

Na začiatok treba povedať, že systém mal pôvodne vychádzať z podkladov systému Brnenskej zoo, pretože tento projekt vznikol s cieľom prvej implementácie a testovania práve tu. Všeobecná funkcionalita systému je popísaná v prvej kapitole, ale pri návrhu interakcie hardvéru, teda turniketov, s databázou bolo potrebné nahliadnuť aj hlbšie do hardvérovej vrstvy. Nakoľko nebola poskytnutá dokumentácia a nebolo umožnené ani nahliadnuť do používaného hardvéru, nedostali sme sa k potrebným informáciám o komunikácií vo vnútri turniketu – o prepojení skenovacieho zariadenia s motorom turniketu, o logike, ktorá je tu využitá. Museli sme teda prísť so svojim riešením a vychádzať pritom zo základných, všeobecne známych funkčných princípov. Ďalej sa v návrhu nachádza úplne nový návrh grafického rozhrania a nový spôsob skenovania a overovania lístkov, popis jednotlivých logických celkov aplikácie a ich funkcionalít. Všeobecne sa jedná o aplikáciu, ktorá je založená na komunikácií užívateľskej a databázovej časti, ktoré si spolu vymieňajú real-time dáta.

Užívateľská časť aplikácie nebude priamo ovládať turniket, rovnako ani API. Z toho vyplýva nutnosť implementovania ovládacej logiky do čítacieho zariadenia umiestneného vo vnútri turniketu, realizujúceho komunikáciu medzi API a hardvérovou vrstvou a požadovanej funkcionality, správnej interakcie s užívateľskými vstupmi. Schéma je znázornená na obr. 2.1. Model sa líši od pôvodne plánovaného návrhu vynechaním middleware komponentu. Problematika je vysvetlená podrobnejšie v podkapitole 2.1.



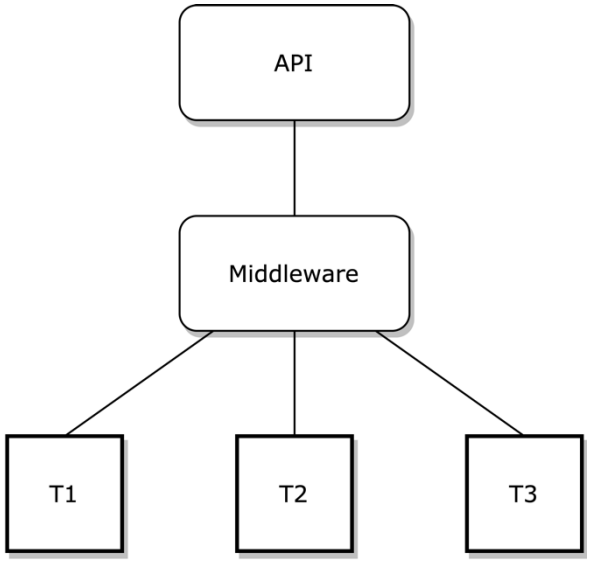
Obr. 2.1 – všeobecný model pokladničného systému

2.1 Návrh komunikácie turniketu s databázou

Táto kapitola sa venuje dvom prístupom návrhu komunikácie hardvérového zariadenia s databázou. Prvý prístup zahŕňa použitie middleware komponentu umiestneného medzi turniketmi a API potrebného pre predávanie dát. Druhý prístup popisuje zmenu návrhu a upustenie od vytvárania middleware komponentu z procesných príčin.

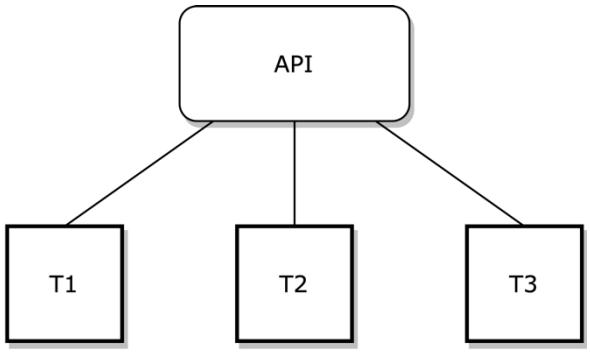
Middleware komponentom sa v pôvodnom koncepte, kedy sa mali prepoužiť aktuálne čítacie hlavice, rozumie lokálny node.js express server bežiaci v pokladničnom zariadení ako ďalšia aplikácia na pozadí spusteného operačného systému. To by znamenalo, že express server sa spustí vždy pri štarte operačného systému na nakonfigurovanom porte. Na tomto porte server počúva a obsluhuje prichádzajúce žiadosti.

Každá požiadavka je spojená práve s jedným turniketom, tzn. je spätá s jeho adresou IP. Po zaregistrovaní požiadavky express serverom sa spustí funkcia, ktorej úlohou bude preposlať dáta zo vstupenky do API. A po prijatí odpovede z API doručiť odpoveď o potvrdení/zamietnutí vstupu na správny turniket, kde sa spustí skript pre ovládanie turniketu v prostredí WinCE. To znamená, že middleware komponent prijíma požiadavky od všetkých turniketov a až potom sú dáta „posunuté“ na API ako je znázornené na obr. 2.2. Postup využívajúci middleware komponent vychádza z predpokladu, že čítacia hlavica je priamo spojená s pokladničným počítačom.



Obr. 2.2 - návrh s middleware komponentom

Pre nedostatok potrebných informácií o komunikácií turniketu a pokladničného PC sa od konceptu s middleware komponentom upustilo a bolo nutné vymyslieť vlastné riešenie, ktoré zachová princíp fungovania systému, ale nebude sa spoliehať na fakt, že turniket je spojený s pokladničným počítačom. Turnikety budú komunikovať s API osobitne, pričom sa ich žiadosti nebudú spracovávať a predávať ďalej v nijakej aplikácií ktorá je umiestnená v strede, ale požiadavky poputujú z turniketu priamo na API a odpoveď zasa odtiaľto späť priamo na turniket. Tým, že z návrhu vypadne middleware komponent (obr. 2.3) je aj dátová cesta v podstate o jeden uzol kratšia, teda systém by mohol byť ešte rýchlejší.

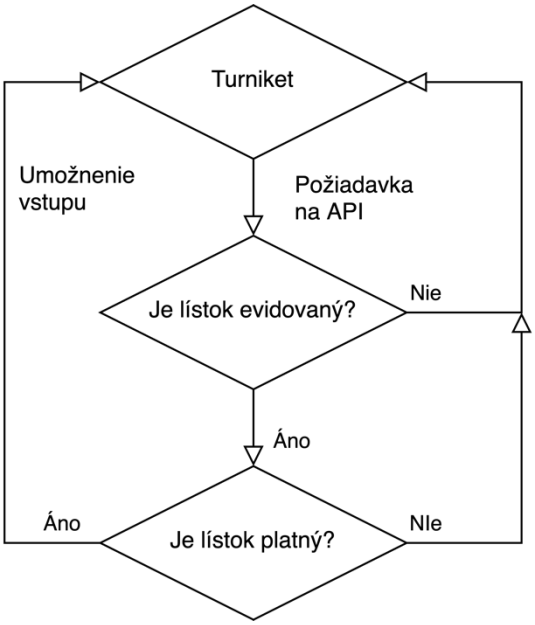


Obr. 2.3 - návrh bez middleware

Ďalej je práca smerovaná s využitím konceptu bez použitia middleware. Proces o pokus komunikácie turniketu s API je zahájený v momente, kedy príde zákazník s lístkom, či už v papierovej alebo digitálnej forme uloženej napríklad v telefóne. Osoba predloží pred skener svoj kód na vstupenke.

Všeobecne platí, že kód vstupenky obsahuje zakódovaný unikátny identifikačný reťazec. Unikátny preto, lebo každý lístok má svoj neduplikovateľný reťazec. Ak sa získa snímka vstupenky, zistí sa, či obsahom snímky je aj kód QR. V kladnom prípade sa dekóduje a až v tomto okamihu pošle turniket požiadavku na API.

Vývojový diagram na obr. 2.4 popisuje postup predávania dát a obsluhy požiadaviek v systéme po zoskenovaní lístka zákazníkom.



Obr. 2.4 - vývojový diagram ovládania turniketov

Lístok/vstupenková karta bude obsahovať zakódované informácie obsiahnuté v čiarovom/QR kóde vo formáte textového reťazca uuid, napr:

```
”0d14826f-4927-4cf3-ae7a-94089”
```

Po oskenovaní lístka turniketom sa id zašifrované v kóde odošle v požiadavke na API. Ak bol lístok naozaj zaplatený, je evidovaný v databáze, a po zaslaní požiadavky na koncovú url v tvare napr. **`/api/ticket/{id},`** sa vráti odpoveď so statusom **`200.`** V ďalšom kroku sa informácia o zvalidovaní lístka pošle späť na čítačku, ktorá otvorí turniket, t.j. návštevníkovi je umožnený vstup.

Ak sa pri validovaní lístka vráti status **`404`** znamená, že požadovaný lístok nebol nájdený v databáze, tým pádom bude označený ako neoriginálny, resp. neplatný, čo bude mať ako dôsledok zamietnutie vstupu, t.j. turniket zostane zablokovaný. Takýmto spôsobom dosiahneme overenie integrity vstupenky a ovládanie vstupných brán.

Obsluha turniketu ako aj proces overovania vstupenky sú podrobnejšie rozpracovné v podkapitolách 2.7 a 2.8.

2.2 Návrh užívateľského rozhrania

Pokladničný systém okrem poskytovania služieb musí obsahovať aj rozhranie, cez ktoré sa bude ovládať. Pri návrhu rozhrania je treba dbať na to, aby sa príliš nelíšilo od

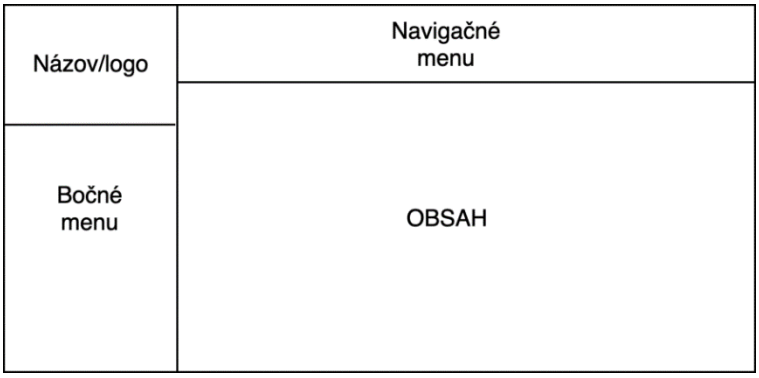
pôvodného a aby si zamestnanci nemuseli zvykať na niečo úplne nové (z hľadiska časovej úspornosti), ale treba mať na zreteli aj inovatívnosť dizajnu, na splnenie požadovanej funkcionality a prehľadnosť spojenú s istou mierou intuitívnosti.

Užívateľské rozhranie bude rozlišovať dva typy užívateľov a to:

- užívateľ typu user,
- užívateľ typu admin,

pričom každému z nich poskytne rovnaký základ užívateľského rozhrania. Odlišnosti systému (medzi užívateľom typu user a admin) vyplývajú z oprávnení týchto rolí. Ako už z názvu vyplýva, admin predstavuje správcu lokálneho pokladničného systému a aby mohol tento systém spravovať potrebuje mať prístup k zvláštnym funkcionalitám aplikácie.

Základný návrh rozdelenia UI, ďalej wireframe, bol navrhnutý v online grafickom editore Figma. Rozloženie komponentov v aplikačnom rozhraní bolo rozdelené na štyri časti, ako je uvedené na obr. 2.5.



Obr. 2.5 - návrh rozdelenia UI

Časť "názov/logo" je čisto reprezentatívna, jedná sa o vyhradené miesto pre logo reprezentujúcu značku aplikácie.

V bočnom menu sa bude nachádzať hlavný zoznam sekcií pokladničného systému. Zoznam bude daný pevne, t.j. nebude sa meniť v závislosti od toho, v akej sekcii sa užívateľ nachádza. Samotná sekcia ako celok, bude pokrývať vždy jednu logickú časť aplikácie, v rámci ktorej bude užívateľovi ponúknutá príslušná interakcia s aplikáciou. Obsah navigačného menu sa naopak bude meniť práve v závislosti od toho, akú sekciu užívateľ aktuálne používa. Časť "OBSAH" je plne závislá na kombinácií vybranej sekcie z bočného menu a záložky z menu navigačného. Podľa prílohy B sú vytvorené štyri hlavné sekcie:

- Obecné,
- Prodej,
- Pokladna,
- Správce,

kde použitie českých názvov vyplýva z dôvodu distribúcie systému do českých

firiem. V sekcii “Obecné” užívateľ nájde základné informácie, prehľady týkajúce sa pokladne, na ktorej pracuje a taktiež záložku “Nastavenia”, v ktorej si bude môcť do istej miery upraviť rozhranie podľa svojich predstáv, zvyklostí.

V oddelení s názvom “Prodej” zamestnanec nájde prehľad všetkých vstupeniek, ktoré je možné zakúpiť. Na základe požiadaviek zákazníka a aktuálneho zoznamu potom môže vystaviť požadovaný druh vstupu.

Panel “Pokladna” slúži k zosumarizovaniu nákupu, podobným štýlom sa využíva napríklad nákupný košík v e-shopoch. Po navolení požadovaných vstupov/akcií/rezervácií sa prejde do tejto sekcie pre kontrolu správnosti vybraných produktov, teda za účelom ukončenia nákupu.

Pre správcu konkrétnej pobočky sa v bočnom paneli nachádza záložka “Správce”, kde bude mať k dispozícií správu a nastavenia celej aplikácie napr. správu užívateľov.

Do navigačného menu bude umiestnený prehľad o tom, aký užívateľ je prihlásený a zároveň rozbaľovacie menu, cez ktoré sa užívateľ bude môcť napríklad odhlásiť.

Pokladničný systém sa bude v štádiu realizácie naďalej vyvíjať, je teda možné, že sa bude finálny koncept realizovaný v rozsahu diplomovej práce do istej miery odlišovať s produkčným. Odlišnosti môžu nastať z hľadiska návrhu UI, ale pochopiteľne aj v rovine funkcionalít, ktoré bude finalizovaná verzia aplikácie ponúkať.

2.3 Popis užívateľskej časti pokladničného systému

Užívateľské rozhranie bolo realizované pomocou knižnice React.js (technológie popísanej v kapitole 1 v kombinácií s typovým jazykom Typescript. Jedná sa o všeobecnú šablónu frontendovej react aplikácie vytvorenú pomocou správcu balíčkov **yarn**, skriptom `yarn create react-app <nazov_aplikacie> --template typescript`. Pri vytváraní boli použité aj ďalšie balíčky, ako napríklad **Formik** na vytváranie formulárov, **Material-ui** za účelom použitia štandardizovaných webových komponentov, ktoré sa týkali najmä vzhľadovej stránky, **formik-material-ui** z dôvodu prepojenia vlastných komponentov s formulármi a zaručenia správnej funkcionality spracovania a odosielania dát na backendovú časť a v neposlednom rade balíček **reach-router**, prostredníctvom ktorého je realizovaná navigácia naprieč celou FE časťou. Aplikácia sa z logického hľadiska skladá z jednotlivých komponentov, ktoré sú všeobecné, a modulov, v ktorých sa jednotlivé komponenty používajú. Jeden logický celok teda predstavuje jeden konkrétny modul, v ktorom sa komponenty používajú podľa potreby, v tomto prípade podľa lokácie, na akej sa užívateľ nachádza.

2.3.1 Komponenty

Štrukturálne sa komponentom rozumie funkcia, ktorá na základe jej vstupných parametrov (props) a logiky zakomponovanej v jej vnútri, vráti časť HTML štruktúry z užívateľského rozhrania, v ktorom sa komponent používa. Jedná sa o variabilné celky, ktoré sú od seba navzájom izolované. V rozsahu FE aplikácie sa jedná napríklad o komponent navigačného menu, bočného menu a formulárových prvkov. Príklad je uvedený nižšie (obr. 2.6).

Obr. 2.6 - príklad komponentu

Z názvu je jasné, že sa jedná o komponent, ktorý bude predstavovať jeden prvok (item) z navigačného menu. Funkcia má typ React.FC (ktorý predstavuje typ funkcionálneho komponentu). Vstupné parametre sú typu `TNavbarItem`. V tomto type sa nachádza aj vstupný parameter *label*. Zo štruktúry je vidieť, že takáto funkcia má ako návratovú hodnotu klasický html `<li>` tag a v ňom štandardizovaný komponent tlačidla, ktorého text zodpovedá práve vstupnému parameter *label*.

Konštanta *classes* zahŕňa objekt štýlov deklarovaných pre tento komponent. Treba si uvedomiť, že konštanta *classes* a atribút tlačidla *classes* sú dve odlišné a spolu nesúvisiace veci.

Štýly sú rozdelené do jednotlivých tried, ktoré si postupne podľa potreby z *classes* objektu vytiahneme a priradíme do atribútu *className*/*classes*. Triedu môžeme priradiť aj podľa logickej podmienky, akou je napríklad zápis `[classes.active]: props.active`. Zápis znamená, že trieda *active* sa priradí na `<li>` element práve vtedy, ak bude hodnota atribútu (vstupného parametru) *active* rovná true.

Na rovnakom princípe sú vytvorené zvyšné komponenty v aplikácií. Niektoré komponenty boli vytvorené úplne „from scratch“ u niektorých sa jedná o úpravu štandardizovaného komponentu z knižnice material-ui.

2.3.2 Moduly

Modul obaľuje jeden logický celok, ktorý zodpovedá obsahu prislúchajúcemu jednej sekcii, ktoré sú popísané v návrhu užívateľského rozhrania (kap. 2.2). Ako prepínač medzi sekciami slúži komponent *Sidebar* - vizualizácia bočného menu. V každom module je použitý rôzny počet a typ komponentov v závislosti od požadovanej funkcionality v konkrétnej sekcii. Funkcionalitu navigácie medzi modulmi zabezpečuje tzv. „Context“, ktorý v tomto prípade slúži ako globálny stav aplikácie. Navigácia v aplikácií bude popísaná v ďalšej podkapitole.

Modulom sa tiež rozumie komponent, ktorý vo svojom vnútri obsahuje ďalšie

komponenty podľa potreby tak, aby bola výsledkom zobrazenia štruktúra jedného logického celku. Modul je komponent top-level úrovne, čo znamená, že nadradené sú mu iba komponenty, ktoré slúžia na poskytnutie prístupu ku globálnemu stavu aplikácie a obsluhe navigácie (viď obr. 2.7).

```
12  export const App: React.FC = () => {
13    return (
14      <AppContextProvider>
15        <LocationProvider>
16          <ThemeProvider theme={theme}>
17            <Router>
18              <Home path="/" />
19              <Tickets path="/tickets" />
20              <CashDesk path="/cash-desk" />
21              <Admin path="/admin" />
22              <Login path="/login" />
23            </Router>
24          </ThemeProvider>
25        </LocationProvider>
26      </AppContextProvider>
27    );
28  };
29
30  export default App;
```

Obr. 2.7 - štruktúra aplikácie

Na obrázku je uvedená celá štruktúra aplikácie. Moduly, o ktorých pojednáva táto podkapitola sú: Home, Tickets, CashDesk, Admin, Login a každý z nich má nastavenú koncovú adresu, na ktorej sa má vykresliť – na tento účel slúži komponent Router, ktorý moduly obaľuje. Ďalej sú modulom nadradené ešte dva komponenty, a to LocationProvider a AppContextProvider, ktoré poskytujú datáciu lokácie a prístup ku globálnemu stavu aplikácie.

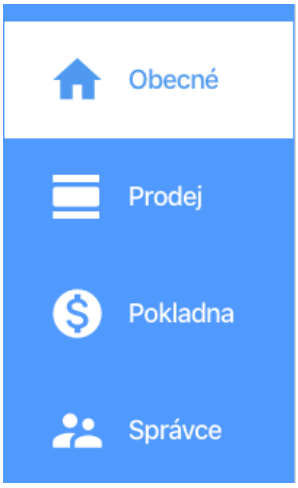
2.3.3 Navigácia

Navigácia aplikácie je realizovaná použitím balíčka **reach-router** v kombinácií s **React.Context**. Táto kapitola nie je venovaná vysvetľovaniu technológií, jej úlohou je popísať čitateľovi postupnosť a princípy funkcionalít, ktoré zabezpečujú navigáciu naprieč aplikáciou.

Základom je použitie komponentov Router a LocationProvider z knižnice reach-router. Komponent Router sa stará o vykresľovanie užívateľského rozhrania podľa toho, na akej url sa užívateľ práve nachádza. Je dôležité zmieniť, že zmena url, ako aj prekreslenie užívateľského rozhrania nevyvolá obnovu stránky, tzv. „refresh“. Práve preto je navigácia a prekresľovanie aplikácie celkom rýchle. Aby bolo jasné, v ktorej sekcii sa užívateľ nachádza, bol použitý komponent LocationProvider, ktorý v svojom stave drží aktuálnu url. Porovnaním tejto url, a url priradenej odkazu v bočnom paneli zistíme, ktorá sekcia je práve zobrazená, teda aktívna, pričom sa zvýrazní.

Vybratím jednej zo sekcií v bočnom paneli (viď obr. 2.8) sa zmení požívateľská url

a to tak, že sa zavolá funkcia na zmenu lokácie, pričom sa lokácia zmení, refresh „stránky“ síce nenastane, ale užívateľské rozhranie sa prekreslí.



Obr. 2.8 – sekcie v bočnom paneli

Zmenia sa hodnoty v navigačnom paneli modulu a budú zobrazené iba aktuálne položky v navigačnom paneli pre modul, do ktorého sa užívateľ prepol. Tu začína byť dôležitá funkcia globálneho stavu aplikácie. Globálny stav aplikácie je reprezentovaný objektom, v ktorom sa nachádza špecifický kľúč „component“. Hodnota uložená v tomto kľúči je závislá od lokácie, v ktorej sa užívateľ nachádza. Pri vykresľovaní každého modulu, je do tohoto kľúča nastavená predvolená hodnota zodpovedajúca prvej položke z navigačného panela. Po kliknutí na iný prvok z navigačného panela, sa do „component“ kľúča uloží hodnota typu **string**, ktorý je obsiahnutý v „key“ parametre štruktúry tohoto prvku (viď obr. 2.9).

```
<NavbarItem
  key={`navbar_item_${key}`}
  label={label}
  onClick={() => {
    dispatch({ type: 'content/set', payload: key });
  }}
  active={key === state.content.component}
/>
```

Obr. 2.9 – zmena globálneho stavu prostredníctvom navigačného panelu

Z demonštračného obrázku vyplýva, že po kliknutí na jedno z tlačidiel v navigačnom paneli sa zavolá funkcia **dispatch**, ktorá slúži na modifikovanie globálneho stavu aplikácie. Táto funkcia má dva parametre: typ a obsah. V tomto konkrétnom prípade sa mení obsah („content“), ktorý sa má užívateľovi zobraziť.

Vykresľovanie komponentov je následne realizované komponent mapou, ktorá je objektom obsahujúcim všetky kľúče, ktoré môžu byť obsiahnuté v navigačnom paneli a ku klúčom hodnoty, ktorými sú jednotlivé komponenty. Hodnota uložená v globálnom stave pod kľúčom „component“ je použitá ako hodnota indexu v komponent mape. Použitie vyzerá takto: `componentMap[state.content.component]`.

2.3.4 Formuláre

Podstatnú časť funkcionality pokladničného systému zabezpečujú práve formuláre. Práve prostredníctvom formulárov sa získavajú dáta, ktoré sa následne posielajú na API pri plnení užívateľských akcií. Jedná sa o klasické html formuláre, s tým, že sa na ich vytvorenie použije špecifický balíček **Formik**, ktorý umožňuje plnú konfigurovateľnosť formulárov v oblasti štruktúry, overovania a samotného odosielania. Štrukturálne sa napríklad definujú typy formulárových polí – či sa jedná o pole, ktorého vstupom je reťazec, číslo, výber z preddefinovaných hodnôt a podobne. V oblasti odosielania dát poskytuje balíček programátorovi voľnosť v používaní vlastných submit funkcií, ktoré sa zavolajú po potvrdení formulára.

Použitie formulárov je jednoduchým a efektívnym riešením práve preto, že veľká časť populácie v dnešnej dobe je zvyknutá používať formuláre na rôzne účely. V tomto prípade je účel formulára jasný – poskytnúť potrebné dáta systému napríklad pri predaji lístka, či vytváraní nového užívateľa.

2.3.5 Overovanie formulárov

Formuláre nemajú za úlohu iba sprostredkovať objekt dát, ktorý sa pošle za účelom vykonania užívateľskej akcie, ale zaistiť aj jeho správnosť. Správnosťou sa rozumie jednak správny formát dát, t.j. zaručenie vyplnenia povinných údajov v konkrétnom formulári ako aj správny typ údajov osobitne.

Funkcionalitu správneho overovania formulárových dát poskytne balíček **yup**, pomocou ktorého sa budú vytvárať tzv. validačné schémy. Validačná schéma má štruktúru objektu, v ktorom sa definuje „shape“ – jeho vnútorný obsah a preň špecifické validačné pravidlá.

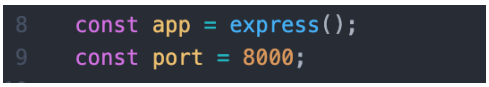
Výhodou **yup** je, že je určený na používanie práve spolu s už spomínaným balíčkom **Formik**. Formulárový komponent obsahuje atribút „validationSchema“ a práve ako jeho hodnota sa vloží Yup overovacia schéma. Spôsob kooperácie validačnej schémy s formulárom je založený na názvoch formulárových polí. Každé formulárové pole má podľa zaužívaného postupu svoj špecifický názov v atribúte **name**, ktorého hodnota je typu string (textový reťazec). Pri vytváraní štruktúry validačnej schémy definujeme pravidlá pre potrebné formulárové polia spôsobom objektového zápisu (**{kľúč: hodnota}**), kde kľúč je zhodný s názvom poľa a hodnota je príslušné validačné pravidlo. Názov formulárového poľa a kľúč s rovnakým názvom v schéme sa automaticky „párujú“ a konkrétnemu poľu sa nastaví overovacie pravidlo. Príklad implementácie validačnej schémy je na obr. 2.10.

Obr. 2.10 - príklad validačnej schémy

Ako je z obrázku vidieť aj chybová hláška je plne editovateľná. Yup taktiež ponúka nastavenie locale, ktoré umožňuje použiť základnú jazykovú mutáciu na špecifické chybové hlášky. Balíček Yup umožňuje aj vytváranie a používanie vlastných overovacích funkcii. Vlastná overovacia funkcia je použitá vo validačnom pravidle, ktoré kontroluje správnosť vyplnenia poľa **number_of_entries**. Ak sa používa vlastná funkcia pre vytvorenie validačného pravidla treba dbať na to, aby návratový typ tejto metódy bol true, false alebo ValidationError.

2.4 Popis serverovej časti aplikácie

Serverová časť aplikácie bola vytvorená za použitia technológie node.js. Jedná sa o serverovú štruktúru. Na vytvorenie servera bol použitý balíček **express** a na komunikáciu s databázou balíček **knex**. Najskôr bolo potrebné lokálne bežiaci server vytvoriť a nakonfigurovať port, na ktorom bude bežať (obr. 2.11).



Obr. 2.11 - vytvorenie lokálneho serveru a nastavenie portu, na ktorom pobeží

Ďalej bolo potrebné nastaviť aj pripojenie k databáze (obr. 2.12). Nastavoval sa typ databáze (client) a zároveň adresa servera, kde bude databáza bežať, meno, heslo a jej názov. Pre testovacie účely pôjde o adresu lokálneho servera, teda 127.0.0.1 (localhost) z čoho vyplýva , že API a databáza budú umiestnené na jednom fyzickom zariadení.

Obr. 2.12 - nastavenie pripojenia k databáze

Serverová časť je rozdelená na dva logické celky a každý obsahuje tri vrstvy. Jeden celok sa venuje operáciám s lístkami, druhý operáciám s užívateľmi. Každý je samostatne popísaný pevne stanoveným koncovou adresou a typom žiadosti. Typy žiadostí môžu byť štandardne GET, POST, PUT, DELETE. Základné koreňové adresy sú v aplikácií deklarované tak, ako je zobrazené na obr. 2.13.

Obr. 2.13 - deklarovanie ciest pre obsluhu užívateľov a lístkov

Všeobecne sú základné cesty pre obsluhu požadovaných funkcionalít **<adresa_domény>/tickets** a **<adresa_domény>/user**. Základnými cestami sú pretože všetky žiadosti pre vykonanie všetkých potrebných akcií týkajúcich sa konkrétnej entity začínajú daným prefixom. Cesta „/tickets“ zabezpečuje manipuláciu s lístkami a „/user“ s užívateľmi.

Aby sa mohli vykonávať požadované operácie s databázou, museli byť vytvorené špecifické koncové cesty pre vykonávanie rôznych akcií a obsluhu rôznych typov žiadostí (viď obr. 2.14).

Štruktúru serverovej časti tvoria tri vrstvy:

1. Prvá vrstva je súhrn repozitárových funkcií, ktoré komunikujú priamo s databázou.
2. Druhú vrstvu predstavujú service funkcie, ktoré volajú repozitárové funkcie.
3. Posledná a najvyššou vrstvou sú tzv. kontroléry, ktoré sa spustia pri zavolaní konkrétnej koncovej url a zabezpečujú výmenu prípadne modifikáciu dát v databáze.

Service funkcie sú metódy z príslušnej triedy, pre lístky je to trieda TicketService a pre užívateľov UserService. Každá funkcia vo svojom vnútri volá repozitárovú funkciu, ktorá podľa typu žiadosti vykonáva operácie s databázou. Ak je typ žiadosti GET, z databázy sa vracajú záznamy na základe parametrov poslaných v url. Pri POST žiadosti sa záznam buď vytvára, alebo vracia, závisí od koncovej url a parametrov v **body** objekte žiadosti. Typ DELETE pochopiteľne slúži na mazanie záznamov.

```
/** Creating ticket */
router.post("/", async (req, res) => {
  const knex = getKnex(req);

  try {
    const ticketService = new TicketService(knex);
    const ticketId = await ticketService.create(req.body);

    sendQRToMail(ticketId);

    res.send(`Ticket was successfully created with id: ${ticketId}`);
  } catch (err) {
    res.send(`An error has occured: ${err}`);
  }
});
```

Obr. 2.14 - funkcia pre obsluhu POST žiadosti na konkrétnej url

Na obrázku je znázornená funkcia pre obsluhu POST žiadosti (z vrstvy kontrolérov). Táto funkcia sa spustí automaticky po zavolaní ${**api**}**/tickets**. Vo vnútri tela funkcie sa vytvorí nová inštancia triedy TicketService, ktorá disponuje funkciami charakteristickými pre manipuláciu so vstupenkami. Pri úspešnom prijatí platných dát z užívateľského rozhrania sa pri použití tejto koncovej url zavolá funkcia **create** so vstupnými parametrami **body** objektu žiadosti a tá v sebe zavolá repozitárovu funkciu s rovnakým názvom – v jej vnútri je definované ako sa dané dáta majú vložiť do databázy. Dáta sa do databázy vkladajú pretože sa jedná o proces vytvárania entinty (vytvorenia záznamu).

2.4.1 Vytvorenie vstupenky

Proces vytvorenia vstupenky začína v momente, kedy operátor/zamestnanec/pokladník vyberie nejakú vstupenk, ktorá sa má predať. Ak svoj výber potvrdí, automaticky sa z užívateľského rozhrania vytvorí požiadavka na koncovú url ${**api**}**/tickets** (viď obr. 2.14), kde **api** parameter predstavuje IP adresu, na ktorej beží API. (Pre vývojové účely má parameter hodnotu http://localhost:8000).

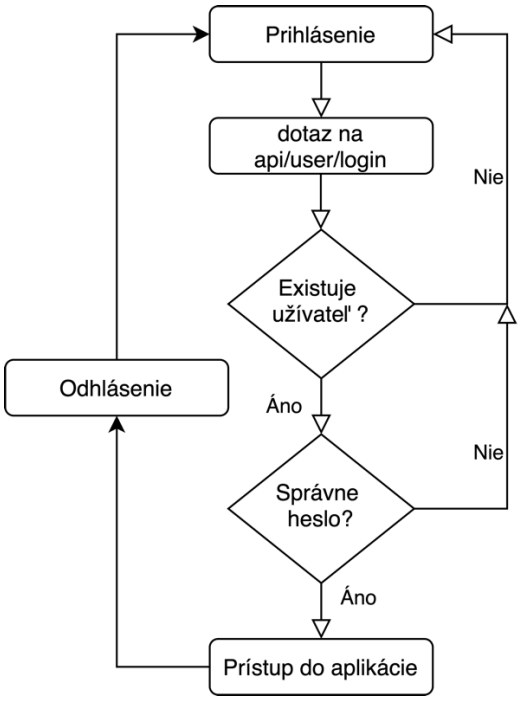
Po zaregistrovaní požiadavky sa spustí funkcia pre vykonanie akcie vytvorenia nového lístka a zároveň sa vygeneruje unikátne id. Toto id sa spolu s ďalšími údajmi, ktoré sú závislé od typu lístka vybranom pri predaji, uloží do databázovej tabuľky s názvom „tickets“.

Pred samotným ukladaním prebehne ešte jeden proces – konverzia id reťazca na QR kód. Práve na tento účel bol použitý balíček **qrcode**, z neho metóda **toDataUrl()** zabezpečí prevod obyčajného textového reťazca na reťazec formátu base64, ktorý predstavuje hotový QR kód.

Identifikačné id je získané tak, že sa vráti ako výsledok volania z funkcie **create**, a po jeho obdržaní je použité ako vstupný parameter do metódy **sendQRToMail**, zabezpečujúcej konverziu id na QR kód spolu s odoslaním kódu do mailovej schránky. Pre tento účel sa používa balíček **nodemailer**.

2.5 Prístup do aplikácie

Užívateľ získa prístup do aplikácie po úspešnom prihlásení cez prihlasovací formulár. Vyplní meno a heslo, tieto údaje sa pošlú na API a v prípade, že server v databáze nájde záznam zhodný so zadaným prihlasovacím menom, porovná zadané heslo s heslom uloženým v databáze. Ak sa tieto heslá zhodujú užívateľské rozhranie sa presmeruje na domovskú stránku a do lokálneho úložiska v prehliadači (localStorage) sa uložia dáta o prihlásenom užívateľovi. Jedným z údajov bude rola – typ užívateľa, teda či je prihlásený užívateľ typu user alebo admin. Na základe toho sa prispôsobia možnosti a operácie, ktoré môže vykonávať prostredníctvom užívateľského rozhrania. Dáta ostanú uložené až do odhlásenia. Postup pri prihlasovaní užívateľa je zobrazený na obr. 2.15.



Obr. 2.15 - cyklus prihlásenia užívateľa

Ak sa užívateľ pokúsi prihlásiť so zlým heslom, z API sa vráti odpoveď, podľa ktorej sa identifikuje chybová hláška zobrazená v prihlasovacom formulári a užívateľ sa dozvie, že použil zlé heslo.

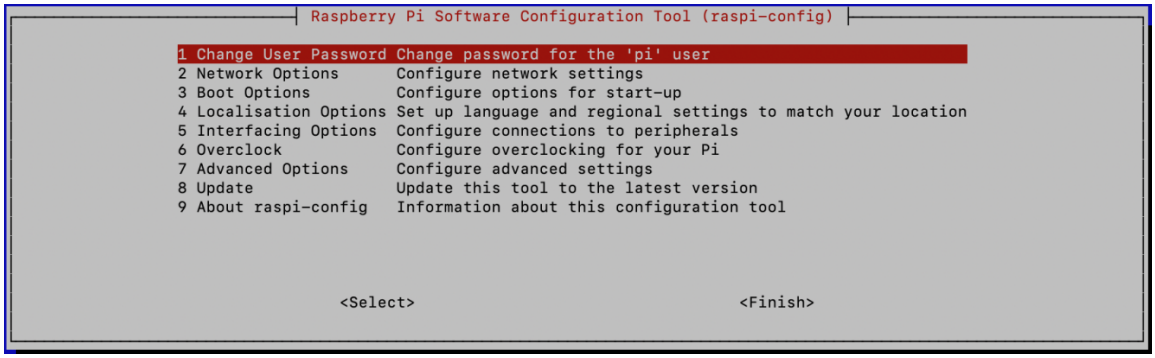
2.6 Nastavenie Raspberry Pi

Skenovanie vstupeniek bude z dôvodu komplikácií pri získavaní dokumentácie fungovať trochu inak, ako bolo pôvodne ustanovené v návrhu. Návrh zahŕňal middleware komponent, ktorý by bežal na pôvodnom turniketovom zariadení – skenovacej hlavici a operačnom systéme, ktorý bol na nej spustený. Pre nedostatok informácií bolo realizované úplne nové, univerzálne vlastné riešenie skenovania QR kódov.

Na skenovanie bol použitý jednodoskový počítač Raspberry Pi, konkrétne model 3B plus. Použitie tohto embedded systému bolo kompromisom jednak časovým ako aj

funkcionálnym. Overovanie platnosti vstupeniek na Windows CE bolo nahradené skenovaním na Raspberry Pi, ktoré natívne podporuje Linux. Konkrétne sa používa Raspbian Buster Lite, ktorý je verzia Buster v minimálnej veľkosti. Systém je načítaný vždy pri spustení Rasberry Pi z SD karty, na ktorej nachádza rozbalený image Raspbianu.

Aby sa dalo k Raspberry Pi pripojiť, je treba pred prvým spustením do súborovej štruktúry na SD kartu vytvoriť súbor s názvom **ssh** (bez prípony). Týmto úkonom bolo povolené pripojenie k RPi z iného zariadenia cez SSH protokol. Po pripojení RPi do lokálnej siete a prihlásení cez základné heslo je odporúčané toto heslo zmeniť a nastaviť cez raspi-config štandardné povolenie SSH protokolu. Raspi-config (obr. 2.16) je konzolová aplikácia pre RPi, v ktorej sa nachádza súhrn všetkých systémových nastavení. Do tejto časti systému môže vstupovať len super užívateľ, preto sa jej spúšťaní pred príkaz raspi-config píše príznak **sudo**.



Obr. 2.16 - raspi-config konzola

RPi je možné pripojiť do siete dvomi spôsobmi. Buď použitím fyzického prenosového média (ETHERNET) alebo bezdrôtovo prostredníctvom Wi-Fi. Ak je zvolená druhý variant, treba nastaviť opäť cez raspi-config konzolu v záložke „Network Options“ správne SSID požadovanej siete, do ktorej sa chceme pripojiť a k nej patriaci správny kľúč. Po dokončení konfigurácie Wi-Fi je potrebné vykonať reboot. Po reboote sa RPi automaticky pripojí do siete. Výhodou Raspberry Pi je, že ak ho užívateľ nastaví raz, pri ďalšom pripojení už pracuje s týmito nastaveniami.

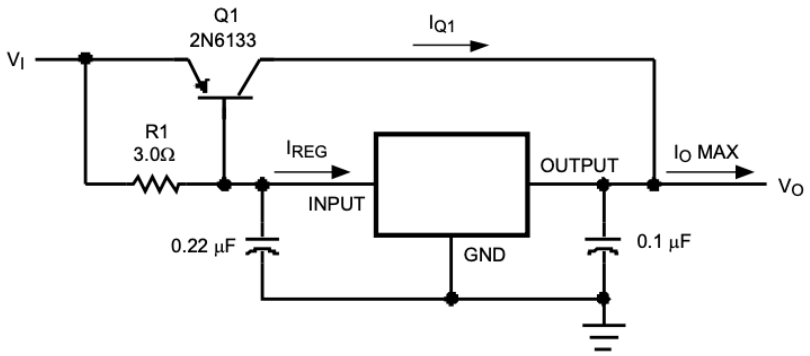
2.7 Napájanie Raspberry Pi

Model 3B+ má k dispozícii vlastný napájací adaptér. V tomto prípade by bolo napájanie z externej siete možné, ale neefektívne, pretože by do turniketu musel byť privedený ďalší zdroj štandardného striedavého napätia 230 VAC, do ktorého by sa zapojil napájací adaptér.

Existuje predpoklad, že Raspberry Pi bude napájané z turniketu. Z tohto predpokladu vyplýva nutnosť navrhnúť taký napájací obvod, ktorý bude vyhovovať špecifikáciám výrobcu tohto jednodoskového počítača. Turniket čerpá energiu z elektrickej siete o napätí 230 V, z bezpečnostných dôvodov je napätie regulované z 230 V na 12/24 V vo vnútri turniketu. Túto hodnotu jednosmerného napätia je potrebné zregulovať na 5 V, za predpokladu, že výstup bude schopný dodať RPi prúd s veľkosťou okolo hodnoty 2,5 A podľa odporúčania výrobcu. Na reguláciu napätia bude použitý obvod LM340A, ktorý má v katalógovom liste uvedenú maximálnu kapacitu výstupného prúdu do 1,5 A

s maximálnou hodnotou vstupného napätia 35 V. Aby bola na výstupe dosiahnutá potrebná hodnota tečúceho prúdu je nutné výstup posilniť [27, 28].

Vyšší prúd na výstupe regulátora dosiahneme rozšírením klasickej aplikačnej schémy o PNP tranzistor, podľa katalógového vzoru zapojenie vysoko prúdového napäťového regulátora (všeobecná schéma uvedená na obr. 2.17).



Obr. 2.17 - všeobecná schéma pre zväčšenie prúdu na výstupe regulátora [27]

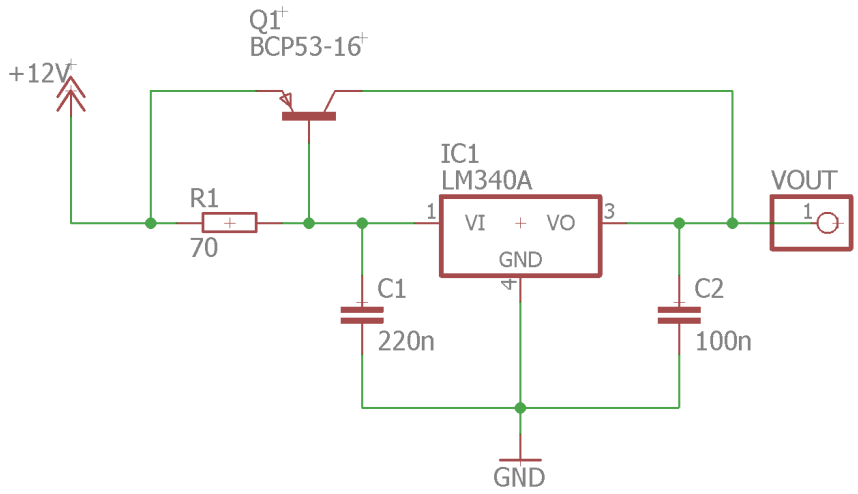
Napájací obvod bude vychádzať z tejto schémy, pričom súčiastky a ich hodnoty budú modifikované tak, aby výstupná prúdová kapacita vyhovovala napájacím nárokom jednodoskového počítača. V prvom rade bude nahradený tranzistor za model BCP53-16, ktorý je takisto PNP tranzistor, ale v SMD prevedení. Maximálny kolektorový prúd tohto tranzistora je 1,5 A. Oproti pôvodným 7 A je poznateľne menší, ale v našom prípade dostačujúci, pretože maximálna prúdová kapacita regulátora je 1,5 A, kapacita nahradeného tranzistora takisto 1,5 A. V súčte teda dostaneme celkovú možnú prúdovú kapacitu na výstupe obvodu 3 A. Kolektorový prúd tranzistorom bol stanovený na hodnotu 1 A, zosilňovací činiteľ tranzistora je hFE je v najhoršom prípade 100. Z toho vyplýva, že najmenší prúd do báze musí mať podľa rovnice 2.1 veľkosť [27, 28]:

$$I_b = \frac{I_c}{\beta} = \frac{1}{100}\ [A], \tag{2.1}$$

kde I_c je prúd kolektorom, I_b je prúd tečúci do báze tranzistora a β je prúdový zosilňovací činiteľ hFE. Z rovnice dostaneme výsledok 10 mA. Veľkosť rezistora R_1 dopočítame tak, aby úbytok napätia na prechode báza-emitor bol 0,7 V. Pre výpočet použijeme formuláciu ohmovho zákona (viď rovnica 2.2):

$$R = \frac{U}{I} = \frac{0{,}7}{0{,}01}\ [\Omega], \tag{2.2}$$

kde U je úbytok napätia na rezistore (rovný napätiu prechodu báza-emitor), I je rovný bázovému prúdu I_b. Veľkosť rezistora bude teda 70 Ω. Po dopočítaní parametrov bude schéma napájacieho obvodu vyzerať podľa obr. 2.18:



Obr. 2.18 - napájací obvod pre Raspberry

2.8 Overenie platnosti vstupeniek

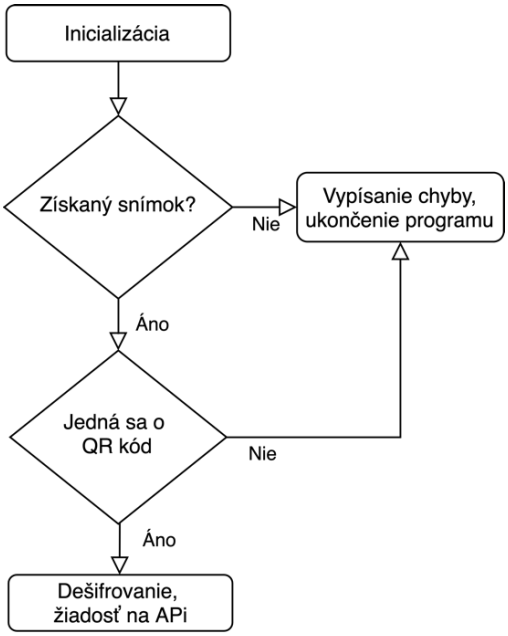
Proces overenia platnosti vstupenky tvoria dve fázy. Prvou je fáza samotného skenovania vstupenky a druhou odosielanie dát na API.

Pre úspešné dokončenie prvej fázy je nutné mať hardvérové zariadenie, ktoré dokáže nasnímať QR kód zo vstupenky. V tomto špecifickom prípade je použitá klasická webkamera s rozlíšením 720p (HD), ktorá je k RPi pripojená cez USB periférium. Aby bolo možné s kamerou pracovať, bolo nutné stiahnuť ovládač **fswebcam**. Nainštalovaním ovládača získame prístup k príkazom na ovládanie kamery, akými sú napríklad zachytenie snímku (obrázku/videa) so špecifickým rozlíšením.

Ďalším krokom pre úspešné zoskenovanie QR kódu zo vstupenky je vytvorenie skriptu obsahujúci algoritmus, ktorý dokáže QR kód rozšifrovať. Skript používa knižnicu **zbarlight**, ktorá implementuje dešifrovacie algoritmy pre rôzne typy skenovateľných čiarových kódov.

Pri spustení algoritmu sa najskôr vždy inicializujú potrebné knižnice, po ich načítaní sa prejde k zachyteniu snímky. Ak je kamera funkčná, snímka sa vytvorí vždy. To znamená, že na snímke získanej pomocou webkamery sa nemusí nachádzať len čiarový kód QR, ale prakticky predmet, prostredie, ktoré sa nachádzajú v okamihu spustenia skriptu pred kamerou. Po úspešnom získaní snímky je potrebné zistiť, či to čo je zachytené na snímke je QR kód, a ak je, následne spustiť dešifrovací algoritmus, ktorého výstupom bude reťazec dát ukrytých v čiarovom kóde. Týmto je ukončená prvá fáza overovania.

Aby bolo možné kontaktovať API, bolo potrebné získať knižnicu, ktorá dokáže obsluhovať http žiadosti. Takouto knižnicou je napr. **requests**. Koncová API url pre overovanie platnosti vstupeniek je už známa **api/tickets/validate/{id}** a žiadosť musí byť typu GET. To znamená, že sa použije **get** metóda z knižnice requests, ktorej parametrom bude naformátovaný url reťazec rozšírený o dáta získané z kódu QR. Algoritmus je popísaný diagramom na obr. 2.19.

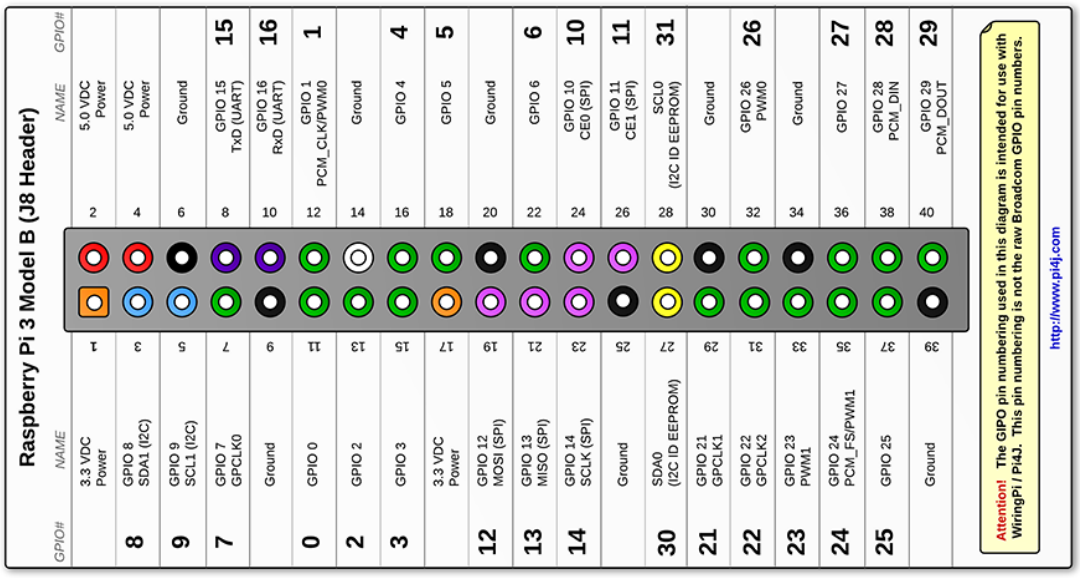


Obr. 2.19 - algoritmus overovania vstupenky

2.9 Ovládanie turniketu

Štandardný stav turniketu je stav zablokovaný. Turniket je držaný elektrickou brzdou, ktorú treba uvoľniť, aby mohol návštevník podniku vojsť do areálu alebo ho v opačnom prípade opustiť. Elektrická brzda sa uvoľňuje privedením signálu na riadiaci modul motora turniketu v závislosti od typu resp. výrobcovi turniketu. V zadaní práce sa nenachádza konkrétny model turniketu, pre ktorý má byť „odblokovanie“ realizované, ďalej sa uvažuje všeobecne.

Raspberry Pi ako embedded systém na doske obsahuje sadu vývodov, ktoré slúžia na rôzne účely (viď obr. 2.20). Z obrázku je zrejmé, že niektoré vývody sú už použité, resp. majú nejaké štandardné mapovanie na periférie, napr. vývody GPIO 12, 13, 14 slúžia na prevádzku periférie SPI. Vo väčšine prípadov je lepšie použiť „voľný“ GPIO vývod, akým je napríklad vývod číslo 0, 2 alebo 3. Otváranie turniketu bude teda prebiehať tak, že po overení vstupenky na API sa vráti výsledok a podľa toho, či bola vstupenka vyhodnotená ako platná sa turniket otvorí – zdvihne sa úroveň na niektorom z vývodov. A v prípade, že bola vstupenka vyhodnotená ako neplatná sa nestane nič a turniket ostane zablokovaný [22].



Obr. 2.20 - pinout RPi model 3B+ [22]

Je nutné podotknúť, že postup ovládania turniketového zariadenia opísaný v tejto podkapitole je všeobecný a podľa potrieb, resp. podľa typu zariadenia, ktoré sa bude ovládať sa môže meniť.

3 DOKUMENTÁCIA POKLADNIČNÉHO SYSTÉMU

Cieľom tejto kapitoly je poskytnúť dokumentáciu k navrhnutému pokladničnému systému, opísať spôsoby používania , zoznámiť užívateľa s jednotlivými procesnými krokmi za účelom správneho používania aplikácie pokladničného systému.

3.1 Základné kroky

Prvým krokom do aplikácie je prihlásenie. Ak používateľ nemá existujúce konto, cez ktoré by sa prihlásil, musí kontaktovať administrátora a ten mu konto vytvorí.

Po prihlásení do aplikácie sa užívateľovi zobrazí užívateľské rozhranie. V ľavej časti obrazovky bude vidieť bočný panel, ktorý bude ponúkať prepnutie (navigáciu) do jednotlivých sekcií aplikácie. V hornej časti obrazovky sa nachádza navigačné menu, cez ktoré sa užívateľ pohybuje medzi logickými celkami v rámci jednej sekcie.

Ako prvá sa zobrazí záložka **„Obecné“**. Táto sekcia bude slúžiť užívateľovi na prehliadanie základných aplikačných informácií a bude prispôsobiteľná požiadavkám zákazníka. Sekcia má tri záložky:

- Nástěnka,
- Informace,
- Nastavení.

V podstate sa jedná o „voľné“ záložky, ktoré nemajú konkrétny zmysel pri predajnej činnosti, ale skôr poskytujú užívateľovi prehľad (**Nástěnka, Informace**), podľa toho, aké budú počiatočné zákaznícke požiadavky. Prostredníctvom karty „**Nastavení**“ si užívateľ bude môcť používateľ prispôsobiť aplikáciu do istej miery podľa svojich predstáv, napríklad mu môže byť umožnená zmena farieb a usporiadanie užívateľského rozhrania.

3.2 Proces vytvárania typov lístkov

Aby bol operátor pokladničného systému schopný predať jednu alebo viacero vstupeniek, musí najskôr vytvoriť inštanciu typu lístka. Pre realizáciu procesu sa operátor musí najskôr navigovať do sekcie **„Prodej“** a v nej sa prepnúť na záložku **„Správa lístků“**, kde sa nachádza formulár na vytváranie vstupeniek. V základnom zobrazení sú k dispozícii dva vstupy – typ a tarifa budúcej vstupenky. Existujú dva typy vstupeniek:

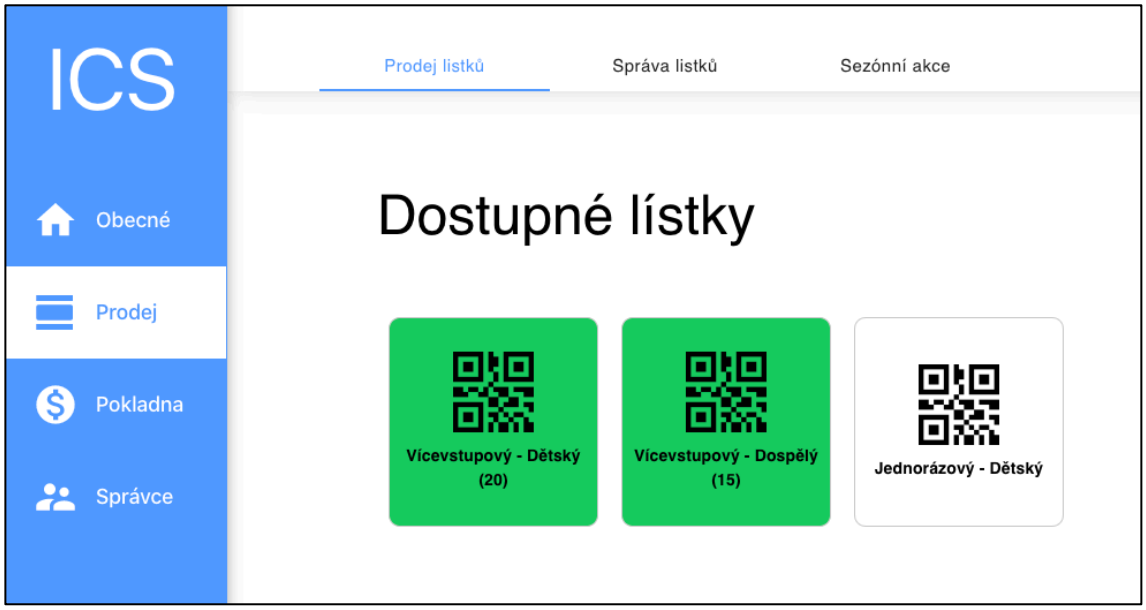
- Jednovstupová,
- Viacvstupová.

A voči dvom typom existuje päť taríf:

- Voľný lístok,
- Detský,
- Študentský,
- Dospelý,

- Seniorský.

Pri vybratí viacvstupového lístka sa užívateľovi zobrazí ešte jeden vstup – údaj o počte vstupov, ktorý je nutné vyplniť. Užívateľský systém ponúka v základe tridsať druhov vstupeniek (päť rôznych jednovstupových, dvadsaťpäť viacvstupových) , pričom opäť treba dodať, že pri produkčnom použití sa bude hľadieť na potreby a požiadavky objednávateľa systému a spektrum lístkov sa môže a nemusí líšiť. Typy lístkov sa pre lepšiu orientáciu budú v užívateľskom rozhraní pre predaj farebne odlišovať (jednovstupové budú mať iné farebné pozadie ako viacvstupové).



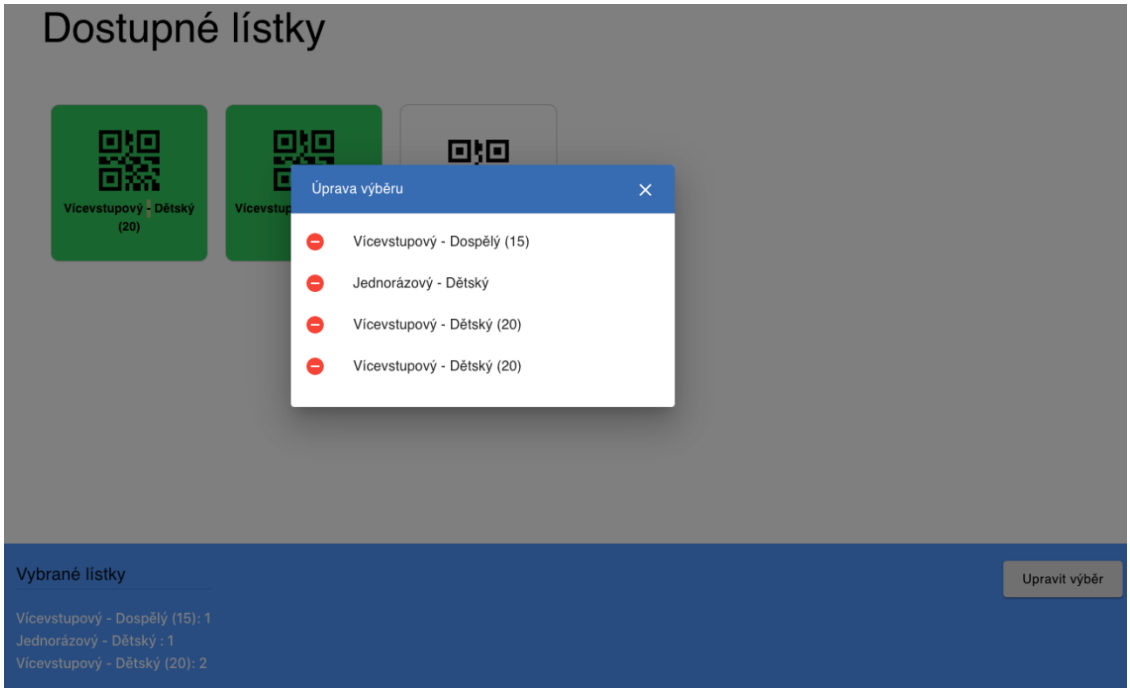
Obr. 3.1 - farebné rozlíšenie typov vstupeniek

Po úspešnom vytvorení vstupenky, sa užívateľovi táto vstupenka zobrazí napravo od formulára v zozname s názvom „**Přehled typů lístků**“, prostredníctvom ktorého bude môcť užívateľ spätne záznam o vstupenke zmazať.

3.3 Vytvorenie záznamu o vstupe

V prípade, že operátor má vytvorené inštancie vstupeniek, budú mu zobrazené v záložke **„Prodej lístků“**. Lístok na predaj sa vyberie kliknutím na dlaždicu lístka a po výbere sa v dolnej časti rozhrania zobrazí súhrn s údajom o type a počte „naklikaných“ lístkov. Súhrn bude ponúkať aj možnosť upraviť výber, teda v prípade nutnosti odobrať lístok z dôvodu pochybenia operátora alebo zmeny rozhodnutia zákazníka, systém intuitívne ponúka možnosť nápravy.

Pokladník môže jednoducho odobrať nechcené vybrané vstupenky a teda nemusí zrušiť celý zoznam a začínať odznova. Užívateľské rozhranie po vybratí lístkov a otvorení dialógu slúžiaceho na editáciu výberu, je zobrazené na obr. 3.2.



Obr. 3.2 - možnosť úpravy vybraných lístkov

V tejto časti užívateľského rozhrania nie je možné predaj potvrdiť, aplikácia funguje na podobnom princípe ako elektronické obchody, v ktorých sa najskôr prejde do košíka a potom platí. Pre dokončenie predaja musí sa musí operátor navigovať do sekcie **„Pokladna“**. Tu nájde operátor opäť výpis uložených vstupeniek do pokladne a končenú sumu, ktorá bude požadovaná od zákazníka na zaplatenie. Výsledná suma sa pochopiteľne bude odvíjať od toho, aké kombinácie typov a taríf lístkov boli vybrané.

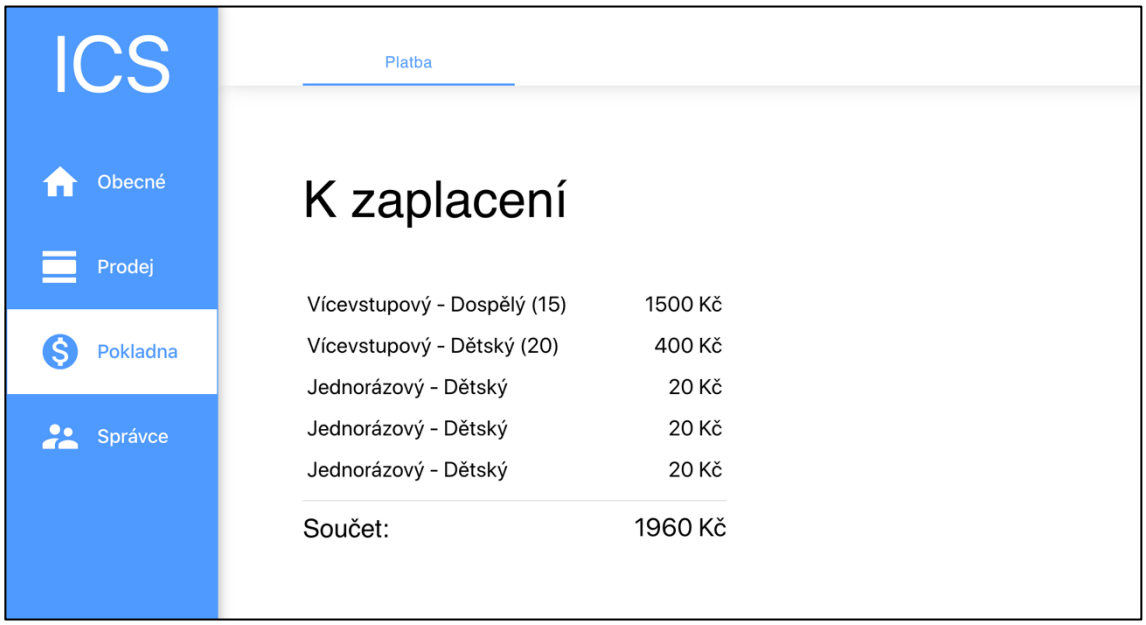
Už pri vytváraní inštancií vstupeniek systém na základe typu, tarify a počtu vstupov priradí lístku jeho nominálnu hodnotu v Kč. Proces prebieha tak, že je stanovená základná cena za jeden vstup pre dospelú hodnotu, napr. 100 Kč. Všeobecne systém počíta podľa rovnice 3.1:

$$Základná\ sadzba * tarif * \ vstupy = cena\ lístku\ [Kč] \qquad (3.1)$$

pre päťvstupový lístok pre dieťa by bola konečná cena lístka:

$$100 * 0{,}2 * 5 = 100\ Kč.$$

Pokladničný systém prepočítava výslednú sumu za všetky vstupenky pri každej editácii zoznamu, teda pri pridaní alebo odobraní lístka (obr. 3.3).



Obr. 3.3 - prepočítavanie výslednej sumy

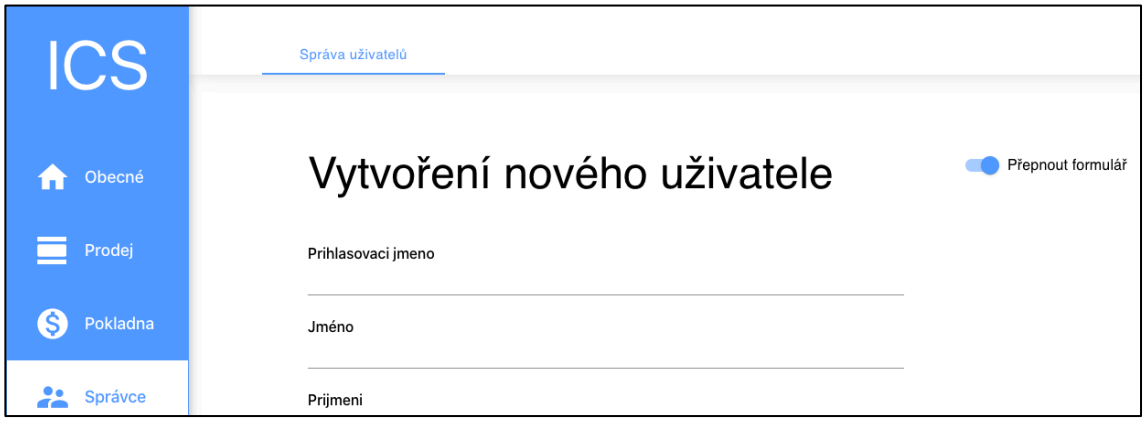
3.4 Vytvorenie a úprava užívateľa

Vytváranie a úprava používateľov sú akcie, ktoré užívateľ vykonáva len z jedného konkrétneho miesta v aplikácií a tým je sekcia „správce“, záložka „správa uživatelů“. To znamená, že bežnému užívateľovi aplikácia vykonávať tieto procesy nedovoľuje. Pre vytvorenie užívateľa sa používa formulár s piatimi vstupmi:

- Přihlašovací jméno,
- Jméno,
- Přijmení,
- Přihlašovací heslo,
- Typ uživatele.

Všetky údaje sú povinné. Na úpravu užívateľov sa používa ten istý formulár, s tým rozdielom, že prvok „Přihlašovací jméno“ nebude už textový vstup, ale výber hodnoty (užívateľského mena) z existujúcich užívateľov. Na základe vybraného používateľa sa hodnoty formulára naplnia dátami vybraného používateľa.

Na prepnutie medzi vytváracím a upravovacím formulárom bude slúžiť prepínač, nachádzajúci sa v pravej časti obrazovky tak, ako je znázornené na obr. 3.4.



Obr. 3.4 - prepínanie formulára

3.5 API referencia

V prípade potreby bude nižšie v texte uvedená API referencia, t.j. zoznam všetkých url, ktoré sa dajú volať vrátane ich popisu. Popis bude v tvare **HTTP metóda: /URL.** Cesty pre manipuláciu s dátami týkajúcich sa vstupeniek:

- **POST: /tickets** - pri zavolaní sa vytvorí záznam o vstupe (+ aktuálne sa pošle aj QR kód do nastaveného mailovej schránky).
- **DELETE: /tickets/:id** - slúži na vymazanie vytvoreného lístka z databázy, parameter **id** je potrebný a je nutné ho poslať ako časť url.
- **DELETE: /tickets/delete/all** - pre vymazanie všetkých záznamov lístkov, zmazanie celej tabuľky s názvom „tickets“.
- **GET: /tickets/validate/:id** - adresa volaná z prostredia RPi po zosnímaní QR kódu zo vstupenky, paramter **id** sa vloží automaticky to url a výsledkom je odpoveď typu Boolean (true | false).
- **POST: /tickets/type** - vytvorenie typu lístka.
- **DELETE: /tickets/type/:id** - vymazanie typu lístka.
- **DELETE: /tickets/type/delete/all** - vymazanie všetkých typov lístkov, vyprázdnenie tabuľky „ticket_types“.

Cesty s metódou **PUT** nebudú používané pri manipulácií dát vstupeniek a ich typov, nakoľko úprava dát už predanej vstupenky/vytvoreného typu lístku nie je žiadúca a mohla by do systému priniesť väčšiu chybu spôsobenú ľudským faktorom. V pokračovaní textu budú uvedené cesty pre operácie s jednotlivými užívateľmi:

- **POST: /user** - koncová url volaná po odoslaní formulára na vytvorenie užívateľa
- **GET: /user** - vráti zoznam všetkých existujúcich užívateľov z tabuľky „users“.
- **PUT: /user/:login** - slúži na úpravu užívateľa, zmenu niektorého z údajov, napr. v prípade potreby zmeny hesla, alebo nahradení zabudnutého hesla a pod. Parameter **login** predstavuje prihlasovacie meno používateľa, ktorého údaje sa majú zmeniť.
- **DELETE: /user/:login** - endpoint slúžiaci na zmazanie užívateľa, ktorého

prihlasovacie meno je zhodné s priloženým parametrom login.

- **DELETE: /user/delete/all** - vymazanie tabuľky „users“ (zmazaní budú všetci užívatelia).
- **POST: /user/signin** - cesta volaná po odoslaní prihlasovacieho formulára, v odpovedi sa vrátia všetky údaje o prihlásenom užívateľovi okrem hesla.
- **GET: /user/:login** - slúži na získanie údajov užívateľa s definovaným prihlasovacím menom.

V tomto prípade má požívanie metódy **PUT** svoje opodstatnenie, ak by sa zakladateľ nového užívateľa (admin) dopustil chyby pri vytváraní užívateľa ako napríklad zle napísané meno, nevyhovujúce prihlasovacie meno, nevyhovujúce heslo a pod., môže svoje pochybenie prostredníctvom práve tejto metódy napraviť.

4 TESTOVANIE

Táto kapitola je venovaná testovaniu systému, teda overeniu základných funkcionalít pokladničného systému. Realizované testy budú v kapitole podložené záznamom žiadosti a získaným výsledkom.

4.1 Prihlásenie

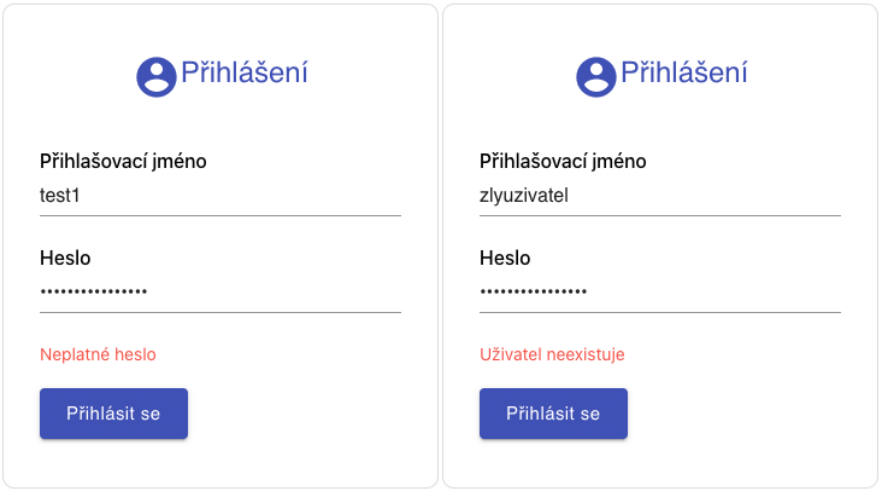
Úspešné prihlasovanie užívateľov je základným predpokladom ďalšieho použitia aplikácie. Na overenie funkčnosti prihlasovania sa musí overiť, či pri spustení aplikácie je nejaký užívateľ prihlásený, či systém dokáže rozpoznať chybné prihlasovacie údaje a pri zadaní správnych údajov užívateľa prihlásiť.

Vytvorený bol testovací užívateľ s dátami podľa obr. 4.1.

	firstName character varying (20)	surName character varying (20)	role character varying (10)	login character varying (20)	password character varying (20)
1	Test	Admin	Admin	test1	test

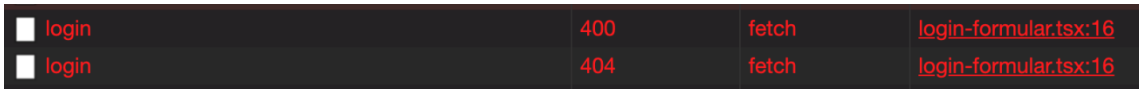
Obr. 4.1 - testovací užívatelia

Pre overenie, či systém dokáže rozoznať zlé prihlasovacie údaje bolo zadané najskôr neplatné heslo a následne prihlasovacie meno (obr. 4.2).



Obr. 4.2 - testovanie neplatných prihlasovacích údajov

Obrázok dokazuje, že API rozoznáva tieto chybne zadané údaje a posiela aj chybovú hlášku, ktorý údaj bol zadaný nesprávne. Ďalším dôkazom je záznam nesprávnych požiadaviek z konzoly webového prehliadača (viď obr. 4.3).



Obr. 4.3 - záznam z google chrome konzoly

Z obrázku je čitateľné, ktorá žiadosť patrí zle zadanému heslu, a ktorá zle zadanému prihlasovaciemu menu. V prvom prípade prišla z API odpoveď so statusom 400 - „Bad request“ (zle zadané heslo) a v druhom 404 – „Not found“ (užívateľ nebol nájdený).

V ďalšom kroku boli zadané relevantné prihlasovacie údaje. Úspešné prihlásenie sa prejaví tým, že aplikácia presmeruje užívateľa dovnútra aplikácie z url „/login“ na url „/“. Po presmerovaní sa v pravom rohu navigačného menu zobrazí meno (údaj firstName) a priezvisko (údaj surName) prihláseného používateľa. Užívateľské rozhranie po prihlásení je zobrazené na obr. 4.4.

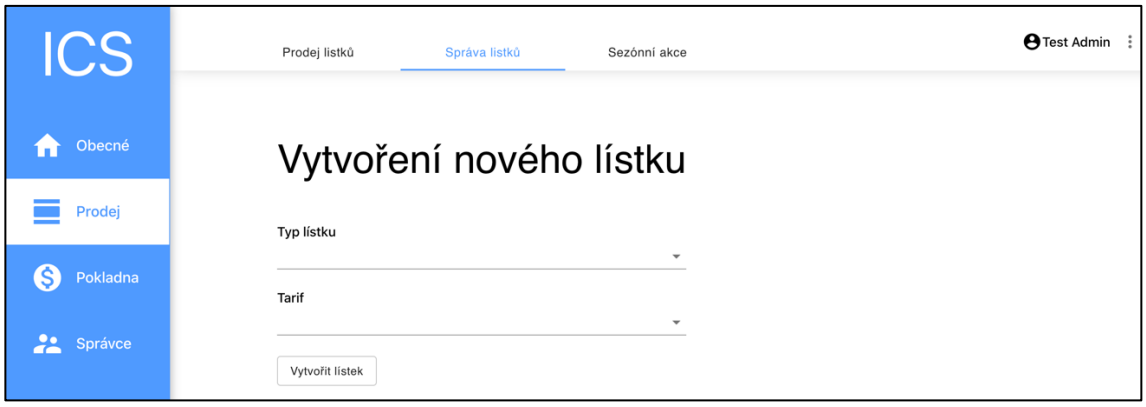
Obr. 4.4 - zobrazenie prihláseného používateľa

Porovnaním údajov z obr. 4.1a z obr. 4.4 vyplýva úspešné prihlásenie. Ďalšia vec, ktorú treba overiť, je rozlišovanie rolí prihláseného užívateľa. Podľa obr. 2.8 by sa mali v bočnom paneli nachádzať štyri položky. Toto platí podľa kapitoly 2.2 iba v prípade, že prihlásená osoba má oprávnenia roly admin. Pri správnej funkcii systému by pri detekovaní prihlásenia užívateľa roly user by sa mala zo zoznamu skryť položka „Správce“. Aby mohla byť overená táto funkcionalita, musel byť pridaný užívateľ typu user. Aktuálne v databáze záznam o dvoch užívateľoch podľa obr. 4.5.

	firstName character varying (20)	surName character varying (20)	role character varying (10)	login character varying (20)	password character varying (20)
1	Test	Admin	Admin	test1	test
2	Test	User	User	test2	tst

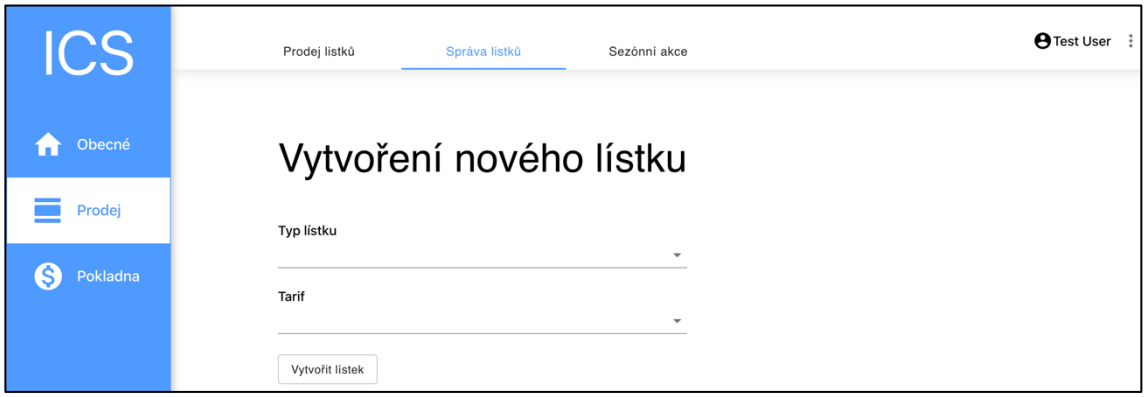
Obr. 4.5 - vytvorenie užívateľa s menšími oprávneniami

Na obr. 4.6 je znázornené užívateľské rozhranie v prípade prihlásenia užívateľa s loginom „test1“.



Obr. 4.6 - prihlásený užívateľ s oprávneniami admin

Na ďalšom obrázku (obr. 4.7) je zmena štruktúry bočného panelu v užívateľskom rozhraní po prihlásení užívateľa login menom „test2“ a rolou user.

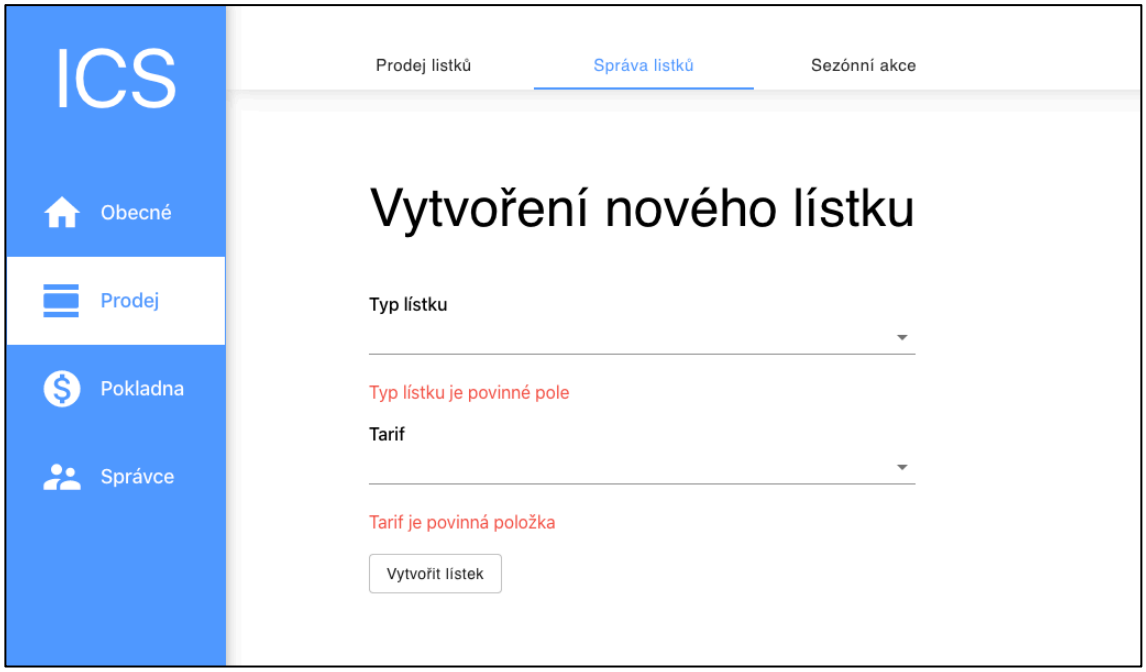


Obr. 4.7 - užívateľské rozhranie pre bežného užívateľa

Porovnaním obrázkov obr. 4.5, obr. 4.6 a obr. 4.7 je dokázaná funkcia rozlišovania užívateľských rolí.

4.2 Proces vytvorenia typu lístka

Cieľom podkapitoly je overiť, že na API sa neodošle žiadosť s neplatným formátom dát, napr. prípad, kedy nie sú vyplnené povinné polia. V najjednoduchšom prípade môže užívateľ vytvoriť lístok na jeden vstup – v takomto prípade sú povinné obe formulárové polia „Typ lístku“ aj „Tarif“. Formulár v počiatočnom stave neobsahuje žiadne dáta. Je teda nutné overiť, že formulár v počiatočnom stave sa nepodarí odoslať. Ukážka overenia vyplnenia polí je na nasledujúcom obrázku.

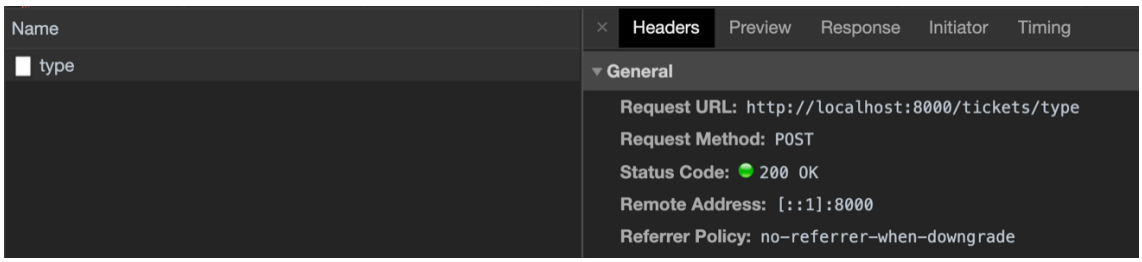


Obr. 4.8 - overenie vyplnenia údajov

Stav znázornený na obr. 4.8 nastal pri pokuse odoslať prázdny formulár. Validačná

schéma patriaca k tomuto formuláru zachytila pokus o odoslanie prázdnych dát a ako dôsledok vrátila nami definované chybné hlášky a zablokovala spustenie funkcie na odoslanie dát z formulára na server.

Ďalším testovacím scenárom už bude odoslanie reálnych platných dát. Na API budú odoslané dáta a sledovaná bude odpoveď. Typ lístka, ktorý sa pokúsime vytvoriť bude teda jednorázový s detskou tarifou.



Obr. 4.9 - dôkaz o úspešnom vytvorení typu lístka

Status 200 znamená, že žiadosť prešla úspešne a na ďalšom obrázku (obr. 4.10) je znázornená aj štruktúra odpovede.

Obr. 4.10 - odpoveď s dátami

Obr. 4.10 znázorňuje odpoveď z API, ktorá obsahuje rovnaké dáta aké boli vložené do databázy. Odoslané boli položky „tariff“ a „type“, v odpovedi sa však nachádza aj „newId“, ktoré predstavuje unikátne id vygenerované na API a pridelené novému typu lístka. Takmer identický proces k procesu vytvárania lístkov je aj proces vytvárania užívateľov. Procesy sú vo svojej podstate zhodné, líšia sa iba v počte a type údajov (teda aj vo validačnom skripte), ktoré sa posielajú na server.

4.3 Overenie vstupenky

Podkapitola je venovaná testovaniu správnej funkcie hardvéru opísaného v kapitole 2.8. zloženého s Raspberry Pi a webkamery, zároveň schopnosti komunikácie s API. Na overenie správnej funkčnosti hardvéru musia byť k dispozícii QR kódy s platným a neplatným identifikačným reťazcom. Pre testovacie účely sa pri vytvorení vstupenky pošle QR kód so zakódovaným (platným) identifikačným reťazcom na mail a takýto QR kód je možné skenovať z obrazovky smartfónu. Skript pre overenie skenovania bude spustený ručne z prostredia RPi, do ktorého sa dá pripojiť prostredníctvom ssh. Po odoslaní požiadavky a prijatí odpovede na danú požiadavku sa prijaté odpoveď vypíše do konzole. Testovať sa v prvom prípade bude neplatný lístok, v druhom prípade platný a porovnávané budú práve výstupy z terminálu.

Obr. 4.8 demonštruje funkcionalitu posielania QR kódu na mail.

Obr. 4.11 - doručenie QR na mail

Po získaní lístka prostredníctvom mailovej schránky bol kód stiahnutý do telefónu a následne z telefónu zosnímaný. Na obr. 4.12 sa nachádza odpoveď z API v prípade skenovanie neplatnej vstupenky, obr. 4.13 demonštruje výsledok skenovania platného kódu. Obidva obrázky predstavujú výstup z terminálu v Raspberry Pi – zobrazená je cieľová adresa a forma žiadosti, id lístka vyčítaného z kódu QR a výsledok platnosti.

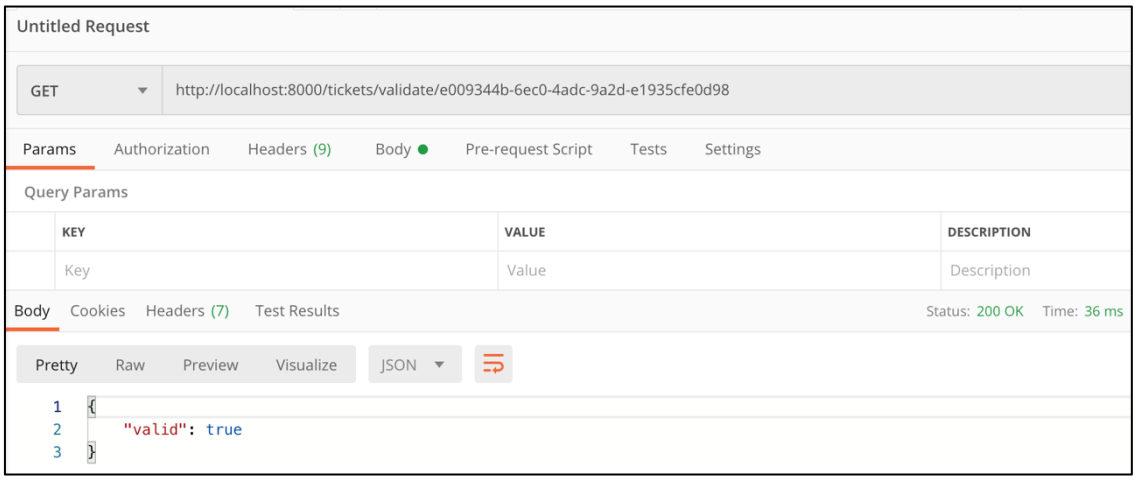
```
pi@raspberrypi:~ $ sudo python reader.py 0
Taking picture..
Picture taken..

Scanning image..

requestUrl http://192.168.1.30:8000/tickets/validate/dc9eb5e2-a8bb-44fa-94f1-1e9ae90aaef4

ticket id: dc9eb5e2-a8bb-44fa-94f1-1e9ae90aaef4
valid: False
```

Obr. 4.12 - skenovanie neplatnej (neexistujúcej) vstupenky

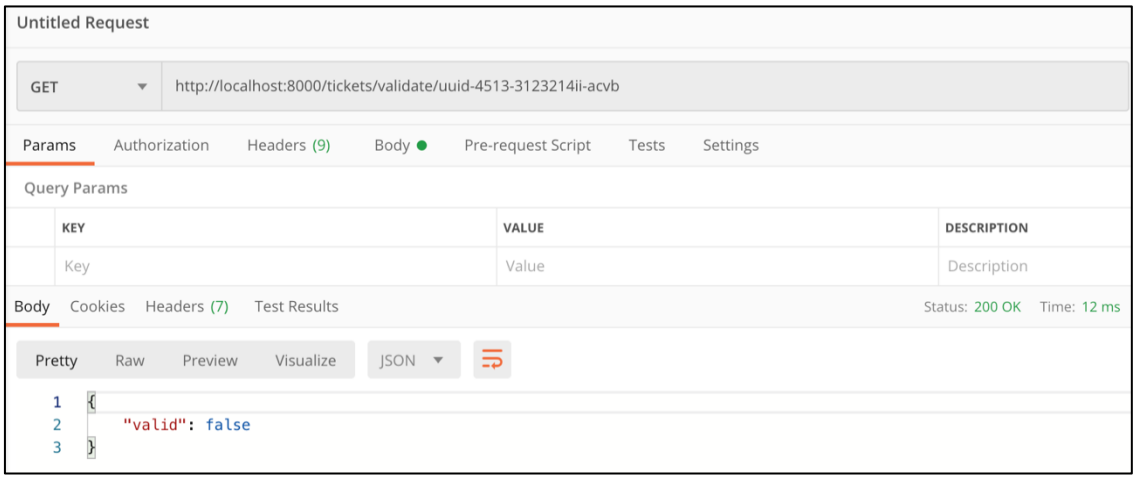
```
pi@raspberrypi:~ $ sudo python reader.py 0
Taking picture..
Picture taken..

Scanning image..

requestUrl http://192.168.1.30:8000/tickets/validate/e009344b-6ec0-4adc-9a2d-e1935cfe0d98

ticket id: e009344b-6ec0-4adc-9a2d-e1935cfe0d98
valid: True
```

Obr. 4.13 - overenie platnej vstupenky

Alternatívne bolo realizované testovanie prostredníctvom aplikácie Postman. Tentokrát sa nebude skenovať lístok z obrazovky smartfónu, ale pošle sa žiadosť s reťazcom id, ktorý sa nachádza v databáze (obr. 4.14) a potom s ľubovoľným reťazcom, ktorý sa v databáze nenachádza a nemá ani rovnaký tvar (obr. 4.15).



Obr. 4.14 - overenie platného id (Postman)



Obr. 4.15 - odoslanie neplatného reťazca (Postman)

Vidíme, že výsledok použitia Postman aplikácie je rovnaký ako v prípade skenovania kódu prostredníctvom hardvéru. Výhodou tejto metódy je zreteľnejší prehľad o žiadosti a štruktúre dát, ktorá sa vráti zo servera ako odpoveď.

Aby bolo dokázané, že v prvom prípade lístok neexistuje a druhý naopak existuje, bola pridaná snímka z databázy, konkrétne z tabuľky patriacej vytvoreným vstupenkám. (v tabuľke sa nachádza iba jeden záznam).

Obr. 4.16 - záznam z databázy

5 ZÁVER

Cieľom práce bolo analyzovať pokladničné systémy, rozobrať ich funkcionalitu, zdokumentovať prevádzkové zariadenia a na základe týchto údajov potom navrhnúť vlastný pokladničný systém, ktorý v rozsahu diplomovej práce implementuje kľúčovú funkcionalitu týchto systémov a poskytne isté funkcionálne rozšírenie na základe požiadaviek zákazníka.

Pri návrhu sa dbalo ako na zachovanie princípov zo systémov, ktoré boli predmetom analýzy, tak aj na inováciu v podobe vlastného grafického návrhu, použitia moderných technológií ako React.js, Node.js, ktoré sú kľúčové pre realizáciu systému.

Návrh a realizácia systému sú popísané v kapitole číslo dva, kde je rozobraná problematika serverovej a užívateľskej časti spolu s návrhom hardvéru použitého na skenovanie vstupeniek s cieľom rozhodovania o ich platnosti. Nachádzajú sa tu vývojové diagramy vysvetľujúce štruktúru jednotlivých logických celkov, postupy použitia pri vytváraní systému.

Kapitola ďalej popisuje a vysvetľuje zmenu v riešení zadania a aj príčinu použitia alternatívneho konceptu oproti pôvodnému, v ktorom mal byť využitý „middleware“ komponent. Pri implementácií pôvodného návrhu sa počítalo s použitím aktuálne používaných skenovacích hlavíc. Pre nedostatok získaných informácií týkajúcich sa interakcie čítacej hlavice s pokladničným serverom, bol navrhnutý nový skenovací hardvér zložený z jednodoskového počítača Raspberry Pi a klasickej desktop webkamery. Keďže sa predpokladá, že RPi bude napájané z turniketu, v kapitole je popísaný aj návrh napájacieho obvodu, ktorého účelom je poskytnúť jednodoskovému počítaču napájací prúd s veľkosťou dostačujúcou na prevádzku počítača vrátane všetkých potrebných periférií.

Základná predloha rozloženia vlastného užívateľského rozhrania bola vytvorená v online grafickom editore Figma, pričom bol kladený dôraz na prehľadnosť, jednoduchosť, logickosť. Koncept grafického návrhu užívateľského rozhrania je súčasťou príloh. Vzhľad konečného stav užívateľského rozhrania vychádza práve z tohto konceptu.

Súčasťou diplomovej práce je aj dokumentácia systému, popísaná v tretej kapitole. Dokumentácia uvádza čitateľa do problematiky orientovania sa v systéme, jeho používania a vlastností.

Záver práce je venovaný testovaniu základných funkcionalít, ako je prihlasovanie užívateľa, rozoznávanie rolí prihláseného používateľa, overenie správneho fungovania typu lístkov a správnej manipulácie s dátami a dokázanie fungovania snímania QR kódov a následného overenia vstupenky.

Výsledkom diplomovej práce je funkčný koncept pokladničného systému, ktorý by v budúcnosti mohol slúžiť po ďalšom vývoji v reálne produkčnej prevádzke alebo aspoň plniť úlohu šablóny, ktorej úpravou sa bude dať docieliť požadovaný výstup.

LITERATÚRA

[1] LEGIC, n-tree solutions Ticketsysteme GmbH [online]. 2019 [cit. 2019-11-15]. Dostupné na: https://www.legic.com/partners/detail/partner/n-tree-solutions-ticketsysteme-gmbh.

[2] NESSY, *Vstupenkové systémy* [online]. [cit. 2019-11-15]. Dostupné na: http://www.nessy.cz/vstupenkove-systemy.

[3] GOTSCHLICH, *Dolomit* [online]. [cit. 2019-11-16]. Dostupné na: https://www.gotschlich.at/files/produkt-infos/TechnischeBeschreibung/Dolomit_Tech_dbl_V2.pdf

[4] GOTSCHLICH, *IKARUS Baden* [online]. [cit. 2019-11-16]. Dostupné na: https://www.gotschlich.at/files/produkt-infos/produktblaetter/EN/IkarusBaden_ProduktblattV3_en.pdf.

[5] N-tree solutions, *Ticket Reader TRX-100* [online]. [cit. 2019-11-18]. Dostupné na: https://www.n-tree.com/fileadmin/platzhirsche/mitglieder/2/22189/Produkte/Lesegeraete/TRX_100-data_sheet-engl.pdf.

[6] Magestore, *What is EPOS system?* [online]. [cit. 2019-11-20]. Dostupné na: https://blog.magestore.com/epos-system.

[7] Hasam, *EPOS 10* [online]. [cit. 2019-11-20]. Dostupné na: http://www.hasam.cz/portals/0/images/hasam/download/EPOS-Popis.pdf.

[8] COMINFO a.s., *REA::TICKET* [online]. [cit. 2019-11-22]. Dostupné na: https://www.cominfo-trade.com/docs_files/ReaTicket_cz.pdf.

[9] PAVLOV, S., BELEVSKY P. *Windows Embedded CE 6.0 Fundamentals* [online]. [cit. 2019-11-23]. Dostupné na: http://staff.ustc.edu.cn/~shizhu/WinCE/winCE6%20Fundamentals.pdf.

[10] V8, *What is V8?* [online]. [cit. 2019-11-22]. Dostupné na: https://v8.dev/.

[11] Open JS Foundation, *About Node.js* [online]. [cit 2019-11-22]. Dostupné na: https://nodejs.org/en/about/.

[12] Open JS Foundation, *Overview of Blocking vs. Non-blocking* [online]. [cit. 2019-11-22]. Dostupné na: https://nodejs.org/en/docs/guides/blocking-vs-non-blocking/.

[13] Strongloop, *What makes Node.js faster than Java?* [online]. [cit. 2019-11-22]. Dostupné na: https://strongloop.com/strongblog/node-js-is-faster-than-java/.

[14] Hackernoon, *Node.js vs PHP: Which is better for web developement?* [online]. Dostupné na: https://hackernoon.com/nodejs-vs-php-which-is-better-for-your-web-development-he7oa24wp.

[15] Facebook Inc., *Introducing JSX* [online]. [cit. 2019-11-23]. Dostupné na: https://reactjs.org/docs/introducing-jsx.html.

[16] Kirill Yusov, *Is React JS fast?* [online]. [cit. 2019-11-23]. Dostupné na: https://jelvix.com/blog/is-react-js-fast.

[17] RESTfulAPI.net, *What is REST* [online]. [cit. 2019-11-25]. Dostupné na: https://restfulapi.net/.

[18] Dynamsoft, *The Comprehensive Guide to 1D and 2D Barcodes* [online]. [cit. 2020-04-10]. Dostupné na: https://blog.dynamsoft.com/insights/the-comprehensive-guide-to-1d-and-2d-barcodes/.

[19] QRcode.com, *Information capacity and version of the QR Code* [online]. [cit. 2020-04-10]. Dostupné na: https://www.qrcode.com/en/about/version.html.

[20] https://www.qrcode.com/en/about/version.html

[21] RASPBERRY PI FOUNDATION, *Products* [online]. [cit. 2020-04-12]. Dostupné na: https://www.raspberrypi.org/products/.

[22] RASPBERRY PI FOUNDATION, *Raspberry Pi 3 Model B* [online]. [cit. 2020-04-12]. Dostupné na: https://www.raspberrypi.org/products/raspberry-pi-3-mode-b/.

[23] The Pi4J Project, *Pin Numbering – Raspberry Pi 3B+* [online]. [cit. 2020-04-13]. Dostupné na: https://pi4j.com/1.2/pins/model-3b-plus-rev1.html.

[24] TechTarget, SearchNetworking, *Ethernet* [online]. [cit. 2020-04-20]. Dostupné na: https://searchnetworking.techtarget.com/definition/Ethernet.

[25] Webopedia, *802.11. IEEE wireless LAN standards* [online]. [cit. 2020-04-20]. Dostupné na: https://www.webopedia.com/TERM/8/802_11.html.

[26] HowStuffWorks, *How WiFi Works,* [online]. [cit. 2020-04-20]. Dostupné na: https://computer.howstuffworks.com/wireless-network1.htm.

[27] Texas Instruments, *LM340, LM340A and LM7805 Family Wide V_{IN} 1.5-A Fixed Voltage Regulators* [online]. [cit. 2020-05-05]. Dostupné na: http://www.ti.com/lit/ds/symlink/lm340.pdf.

[28] RASPBERRY PI FOUNDATION, *Power Supply* [online]. [cit. 2020-05-05]. Dostupné na: https://www.raspberrypi.org/documentation/hardware/raspberrypi/power/README.md.

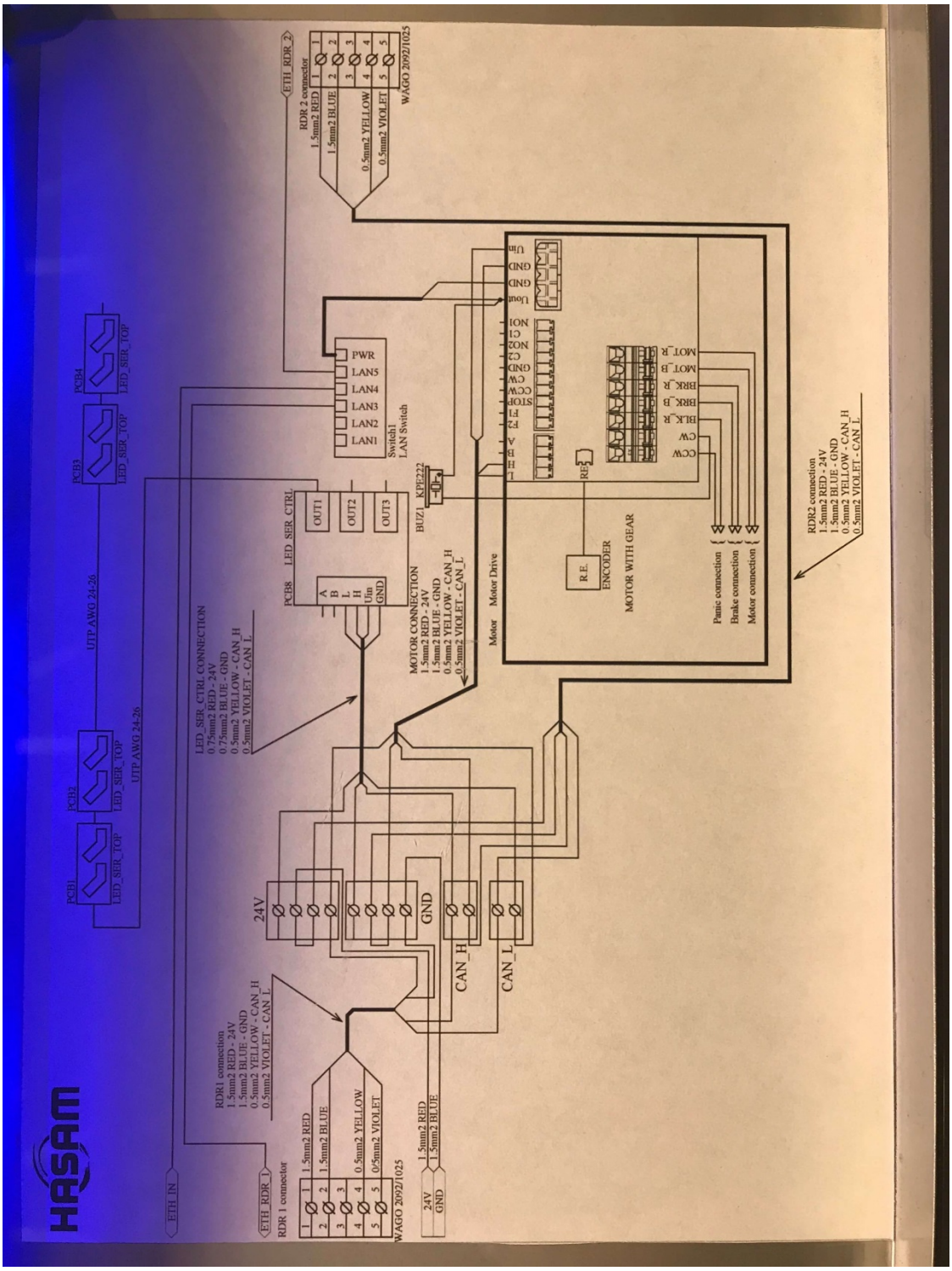
ZOZNAM SYMBOLOV, VELIČÍN A SKRATIEK

API	Application programming interface.
UI	User interface.
RPi	Raspberry Pi.

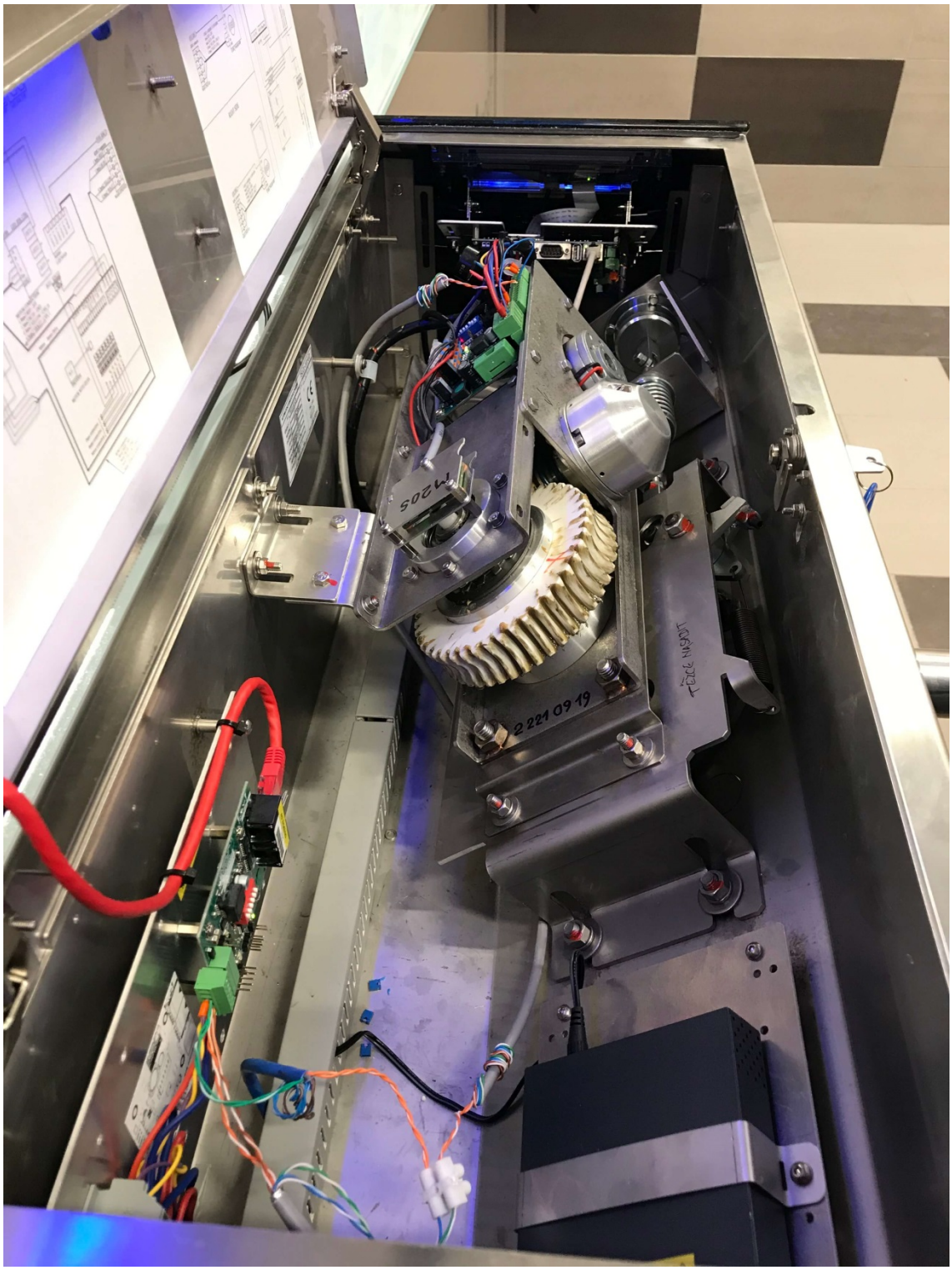
ZOZNAM OBRÁZKOV

A TURNIKET HASAM TS-121-1014

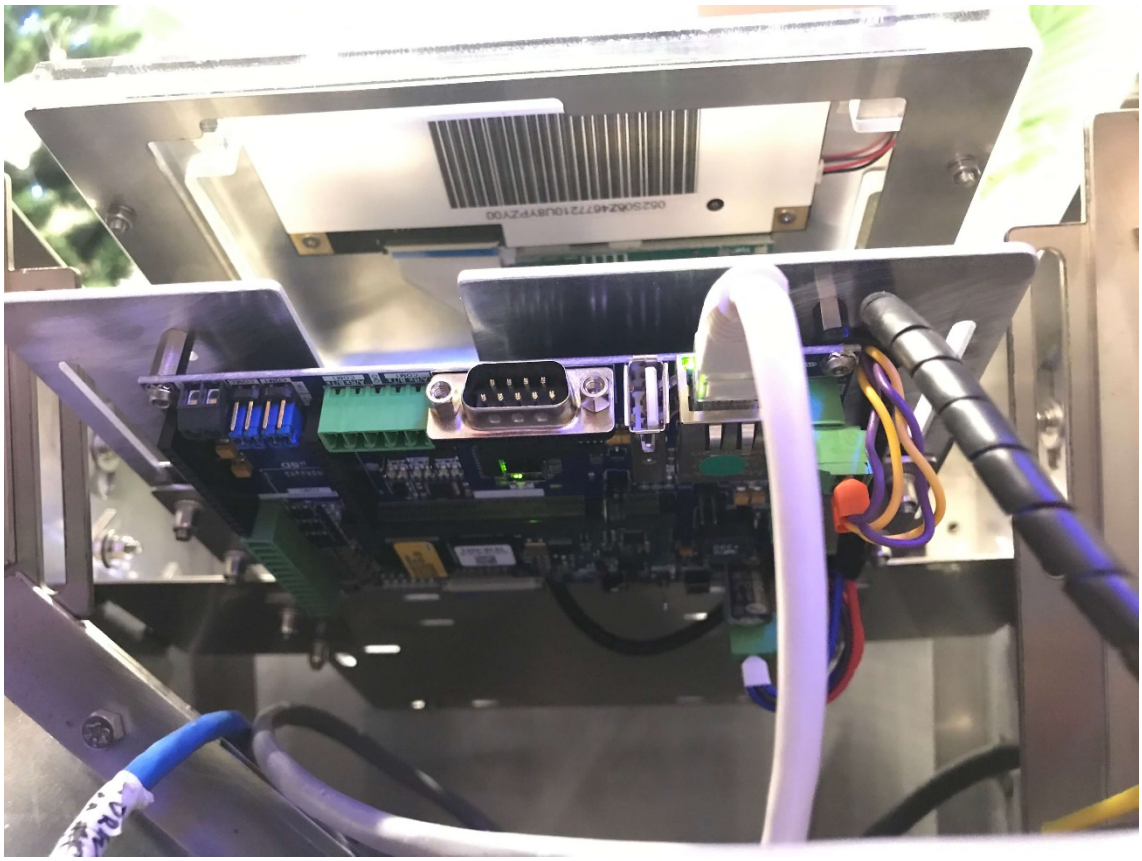
A.1 Obvodové zapojenie turniketu



A.2 Motor turniketu



A.3 Riadiaci počítač vnútri turniketu



B ZÁKLADNÝ NÁVRH UŽÍVATEĽKÉHO ROZHRANIA